RADACS: Resolution Aware Diffusion Accelerator Leveraging Computational Staleness

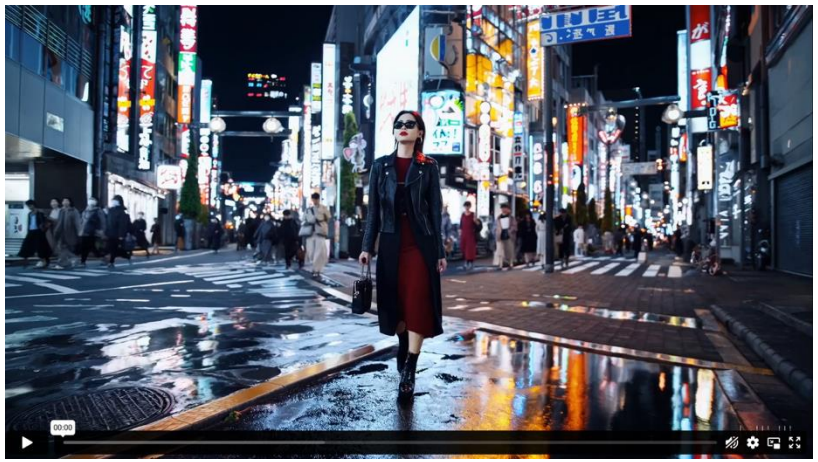
Team Makgong (O. Kwon, K. Oh, H. Yu)

목차

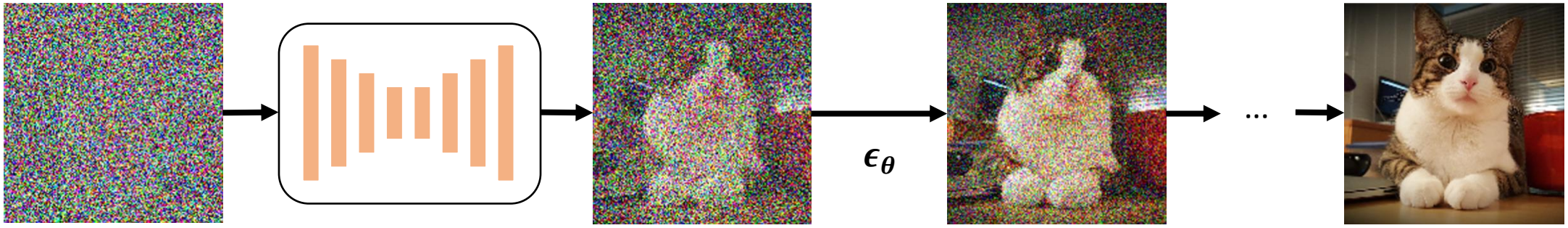
01. Background and Motivation

Diffusion Model

Image Generation



Video Generation



U-Net ϵ_θ

$\mathbf{x}_T$ Noise

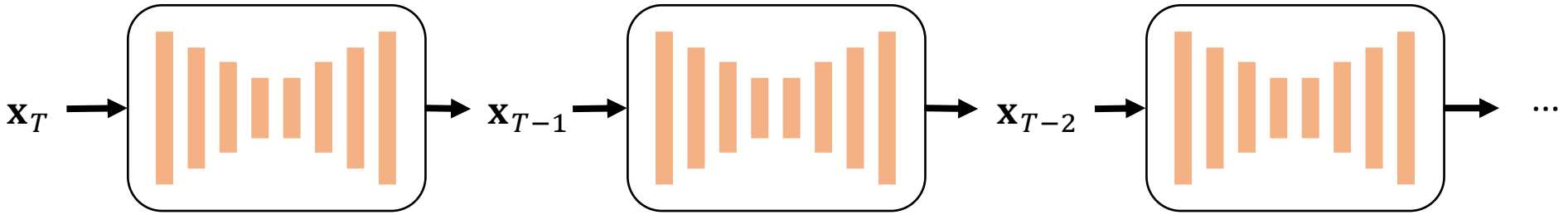
$\mathbf{x}_{T-1}$

$\mathbf{x}_{T-2}$

$\mathbf{x}_0$ Data

Necessity of Diffusion Acceleration

Diffusion passes through U-Net sequentially over multiple timesteps, leading to high latency and difficulty in parallelization.

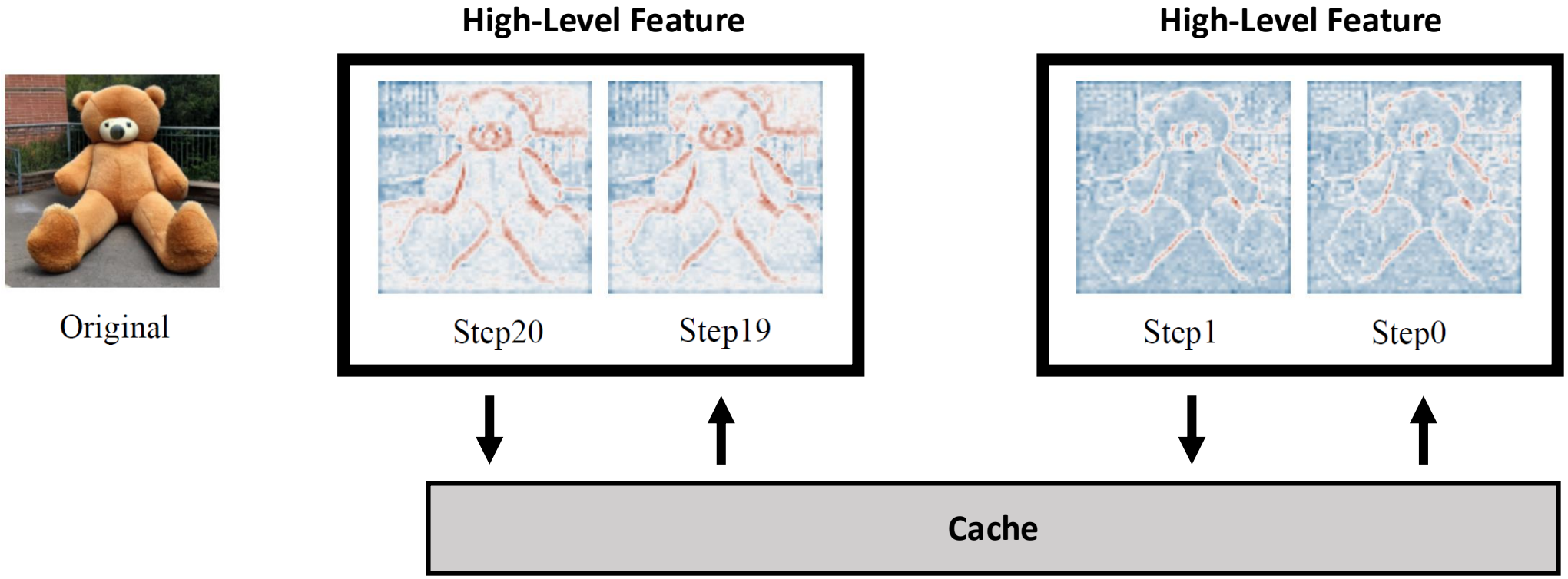


Diffusion Inference on Edge Device

Need for a diffusion accelerator!!

02. Related Works

1. Staleness (1): DeepCache

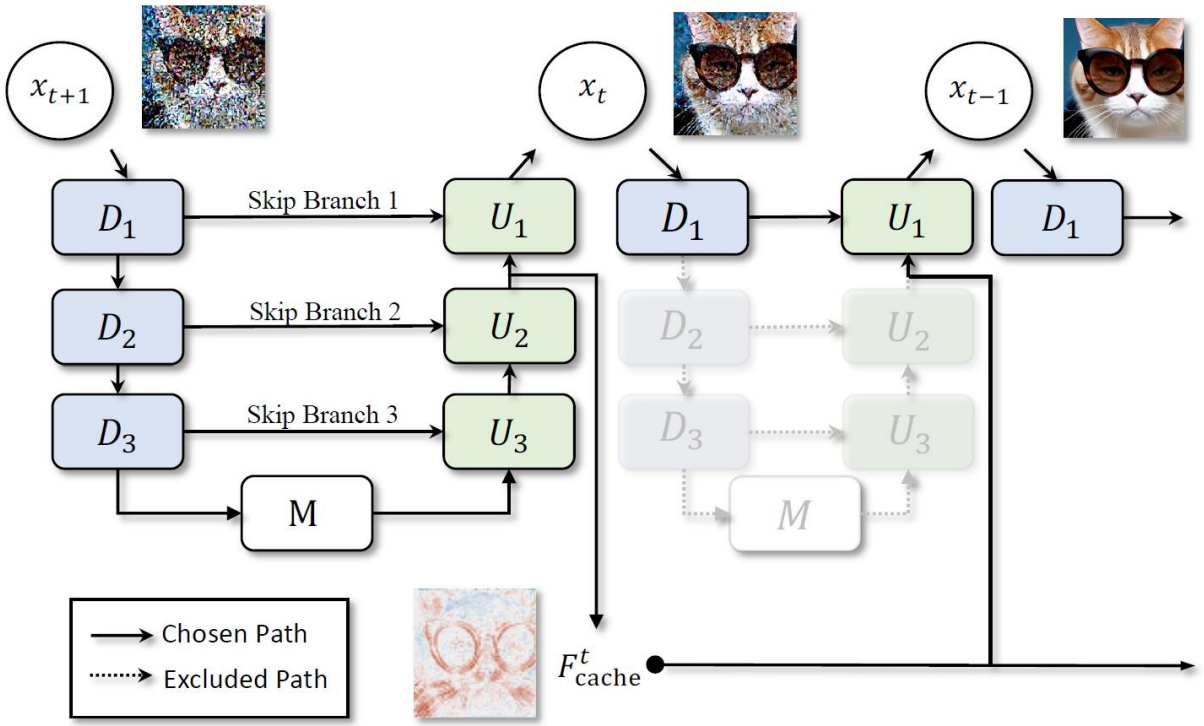


- DeepCache[1] is an algorithm that accelerates diffusion inference without additional training.
- The high-level features of U-Net between adjacent timesteps are highly similar.
- By caching the high-level features from the previous timestep, computational load is reduced.

[1] X. Ma, G. Fang, X. Wang, "Deepcache: Accelerating diffusion models for free," in *CVPR*, 2024.

Related Works

1. Staleness (1): DeepCache



- Low-level feature $D_1^t(\cdot)$
- High-level feature $\boldsymbol{U_2^t}(\cdot)$
- The high-level feature uses cached values from the previous timestep:

$$U_2^{t-1}(\cdot) \approx \boldsymbol{U_2^t}(\cdot)$$

- Acceleration algorithm utilizing staleness.

[1] X. Ma, G. Fang, X. Wang, "Deepcache: Accelerating diffusion models for free," in *CVPR*, 2024.

Related Works

1. Staleness (1): DeepCache

Stable Diffusion v1.5

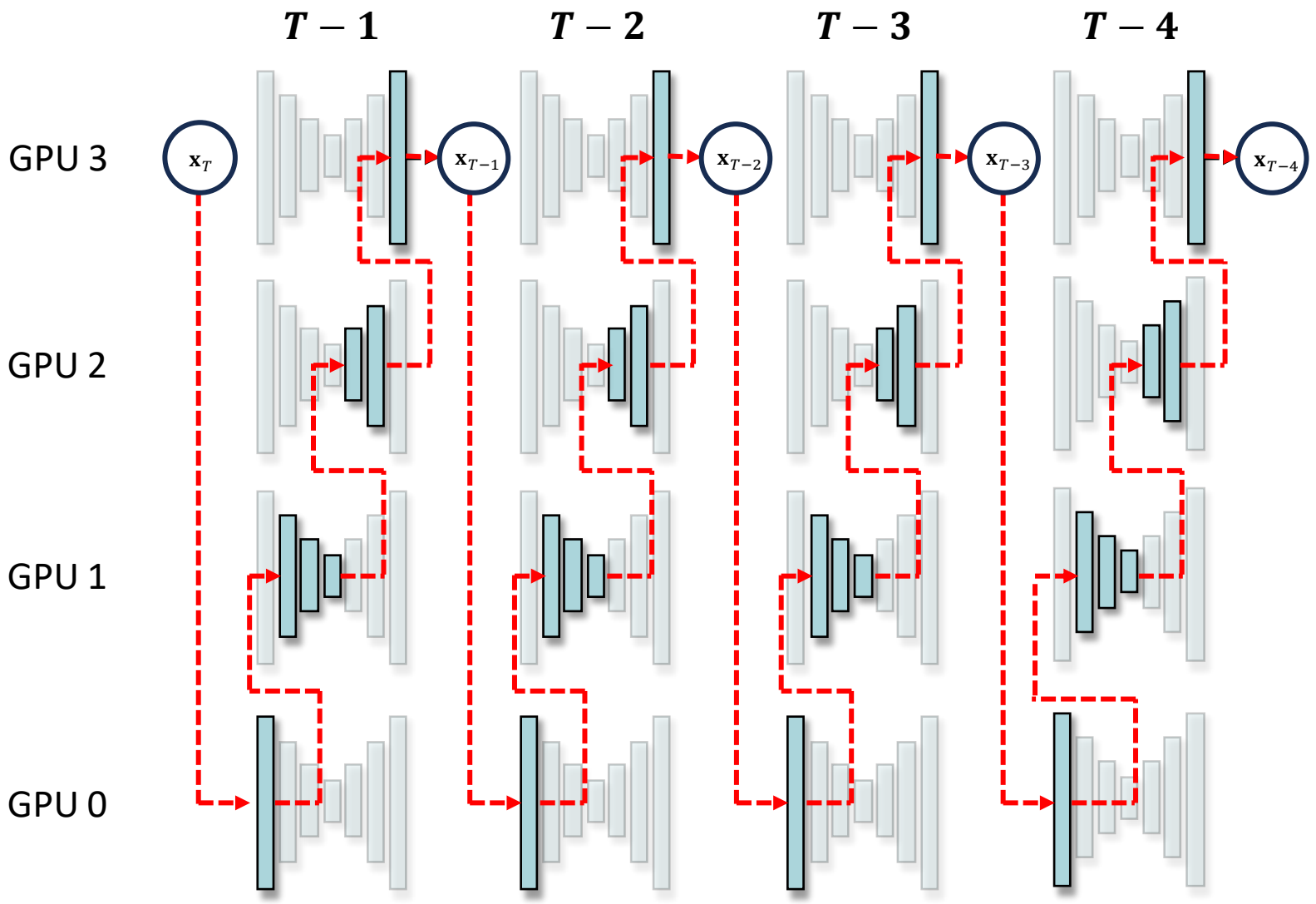
Original

2.3× speedup

DeepCache

[1] X. Ma, G. Fang, X. Wang, "Deepcache: Accelerating diffusion models for free," in *CVPR*, 2024.

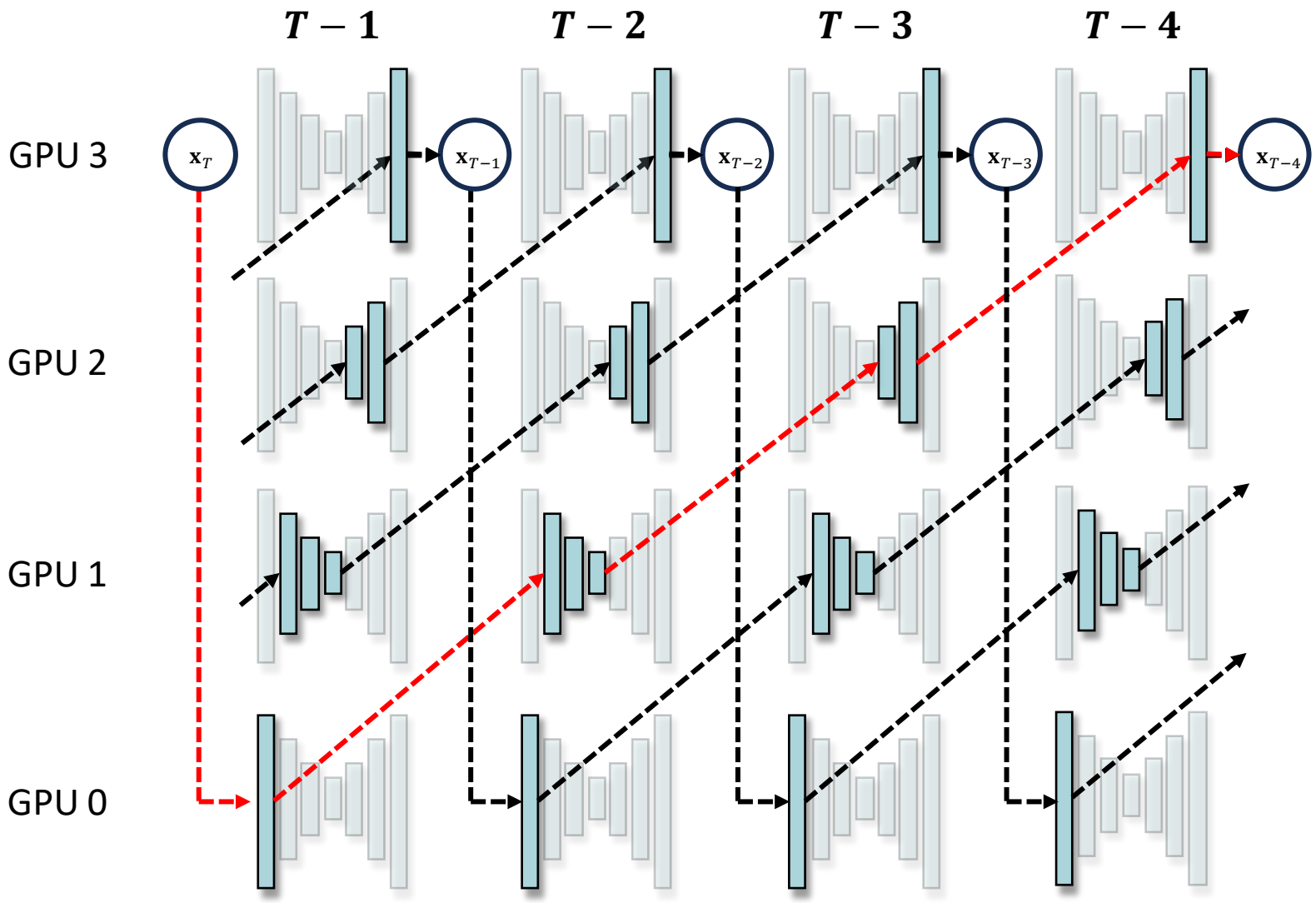
1. Staleness (2): AsyncDiff



- Divide U-Net into sequential components and allocate each to a separate GPU
- High latency occurs as each component must be executed sequentially
- Low GPU utilization and difficult parallelization

[2] Z. Chen, X. Ma, G. Fang, Z. Tan, et al., "AsyncDiff: Parallelizing diffusion models by asynchronous denoising," in *NeurIPS*, 2024.

1. Staleness (2): AsyncDiff



- Remove operation dependencies by leveraging the similarity of features between adjacent timesteps.
- Use the previous timestep's feature as an approximation for the current timestep.
- All GPUs compute different parts of the U-Net in parallel within a single timestep.

[2] Z. Chen, X. Ma, G. Fang, Z. Tan, et al., "AsyncDiff: Parallelizing diffusion models by asynchronous denoising," in *NeurIPS*, 2024.

1. Staleness (2): AsyncDiff

Stable Diffusion XL

Original

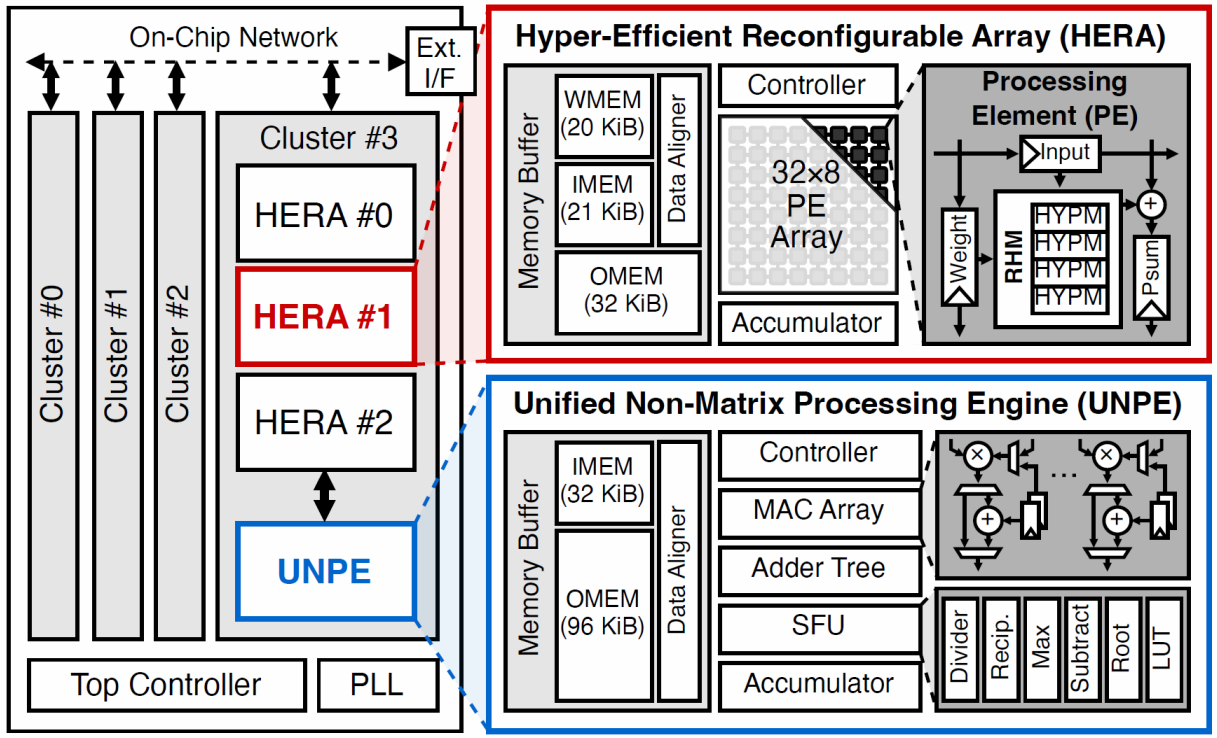
2.8× speedup

AsyncDiff

[2] Z. Chen, X. Ma, G. Fang, Z. Tan, et al., "AsyncDiff: Parallelizing diffusion models by asynchronous denoising," in *NeurIPS*, 2024.

2. Diffusion Accelerator: Picasso

New data types, optimized PE arrays, and non-matrix unit designs for diffusion acceleration.



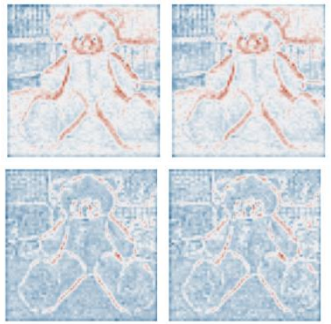
- Hyper-Precision Data Type (HYP8)
- Hyper-Efficient Reconfigurable Array (HERA)
- Unified Non-Matrix Processing Engine (UNPE)

[3] S. Yoo, G. Ko, S. Ham, et al., "Picasso: An Area/Energy-Efficient End-to-End Diffusion Accelerator with Hyper-Precision Data Type," in *ESSERC*, 2024.

RADACS Overview

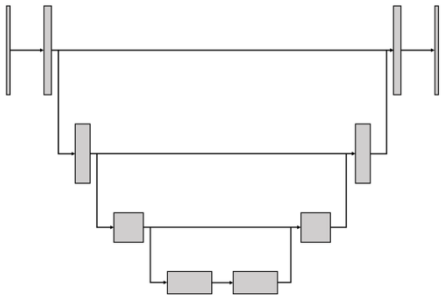
RADACS

1. Utilizing Staleness



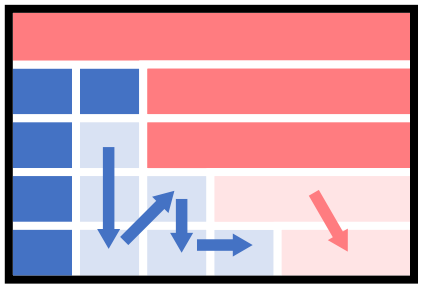
- Apply Computational Staleness to hardware design.

2. Leveraging U-Net's Structural Characteristics



- Design different Matrix Multiplication Units reflecting the size of the feature map.

3. Solutions for Various Workloads



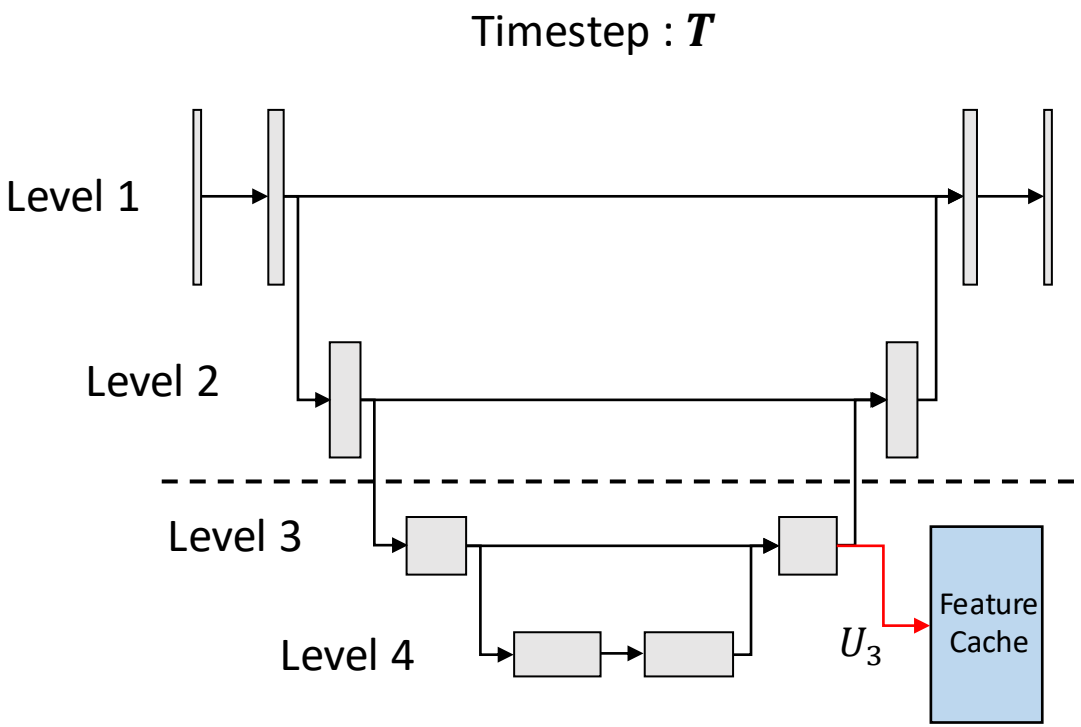
- Propose various scheduling algorithms, considering the number of batches.

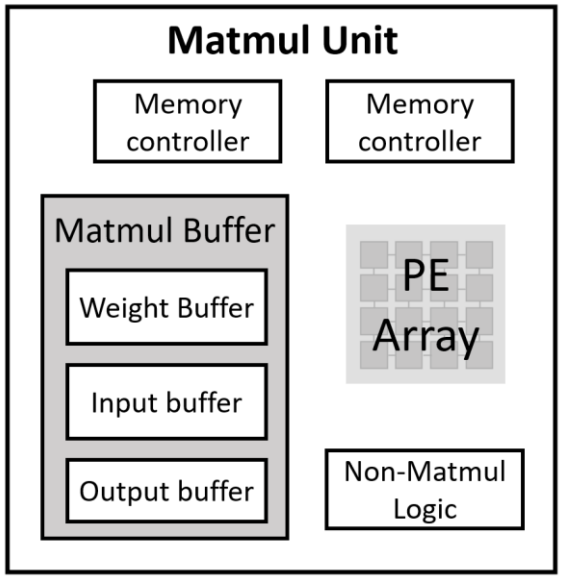
03 . Proposed Ideas

Proposed Ideas

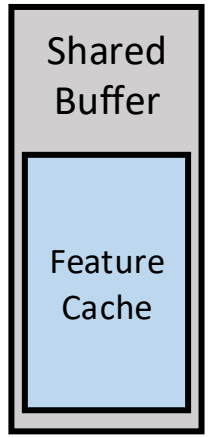
Feature Cache

At timestep T, features are stored in cache.
At timestep $T-1$, stored features are loaded.





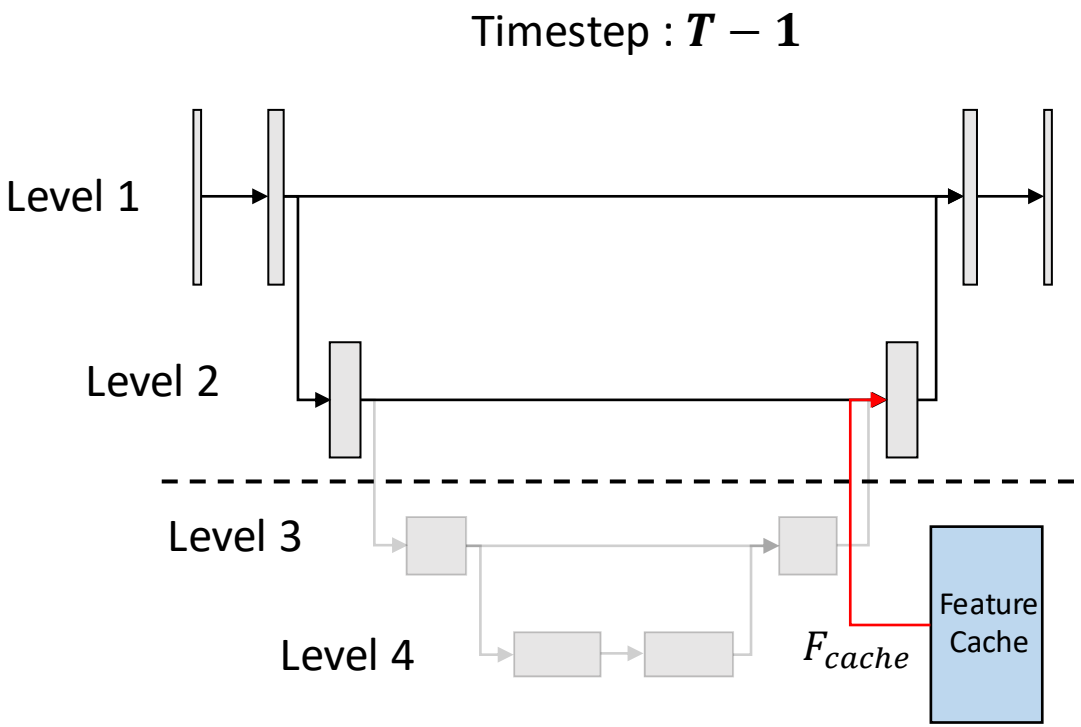
Store U_3 at t
$(U_3 \rightarrow F_{cache})$

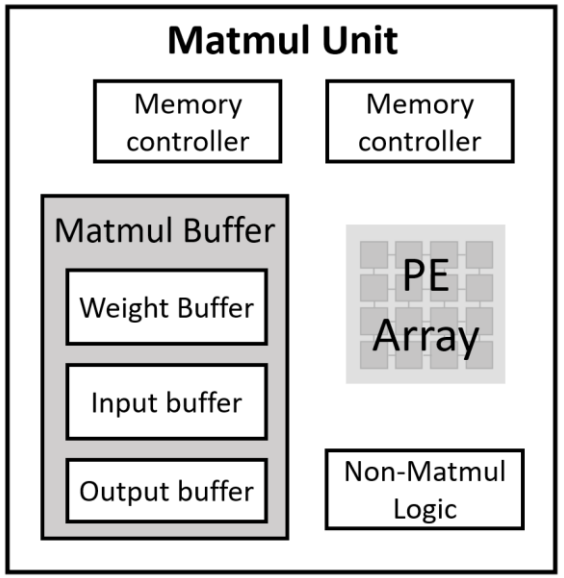


Proposed Ideas

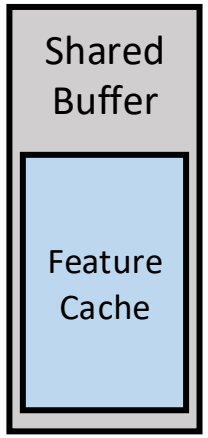
Feature Cache

At timestep T, features are stored in cache.
At timestep $T-1$, stored features are loaded.





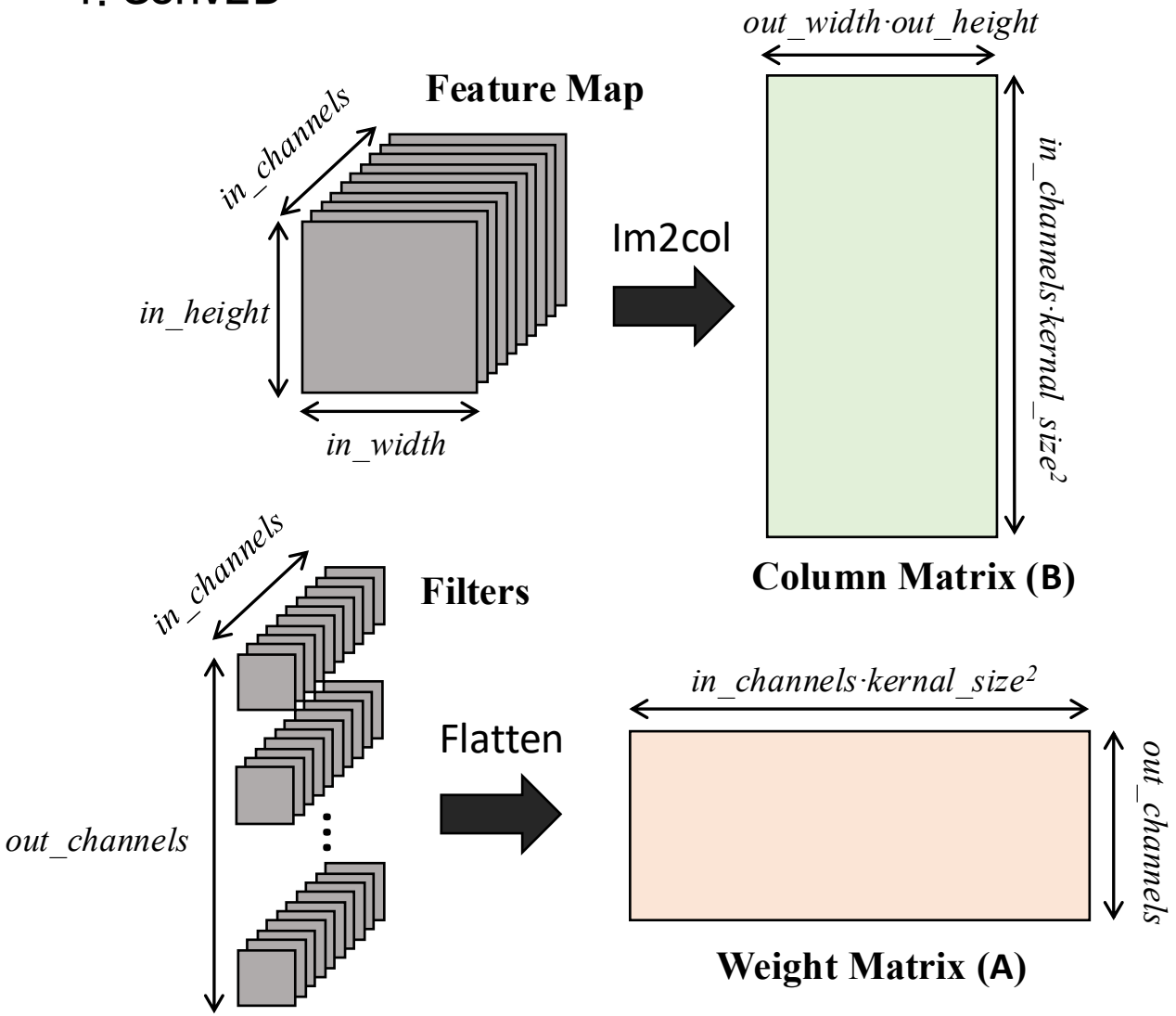
Load F_{cache} at $t-1$

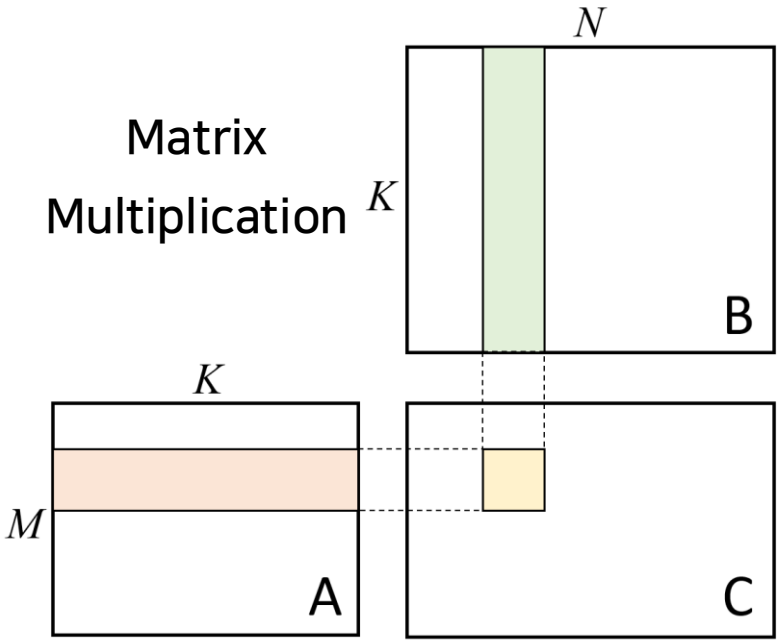


Proposed Ideas

Characteristics of Diffusion Computation

1. Conv2D



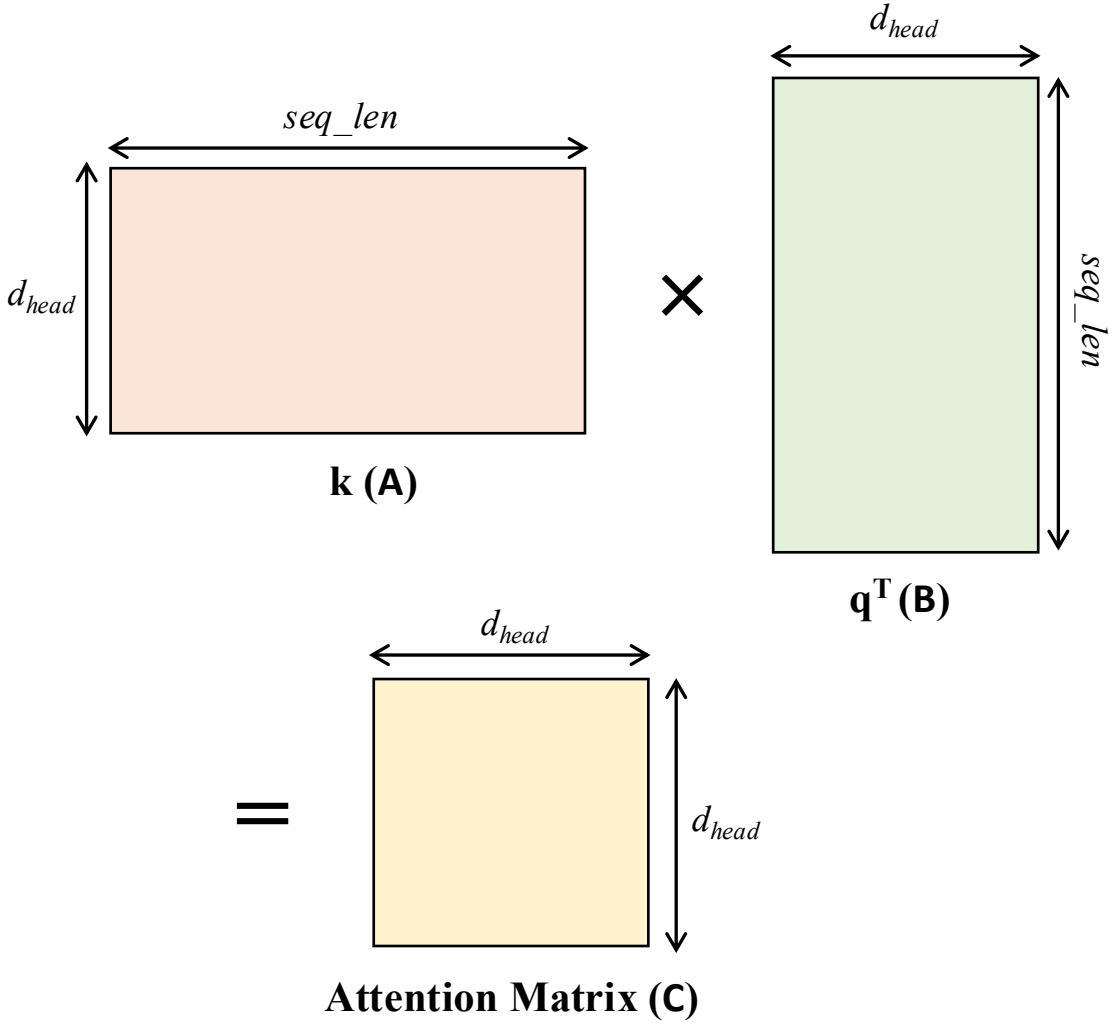


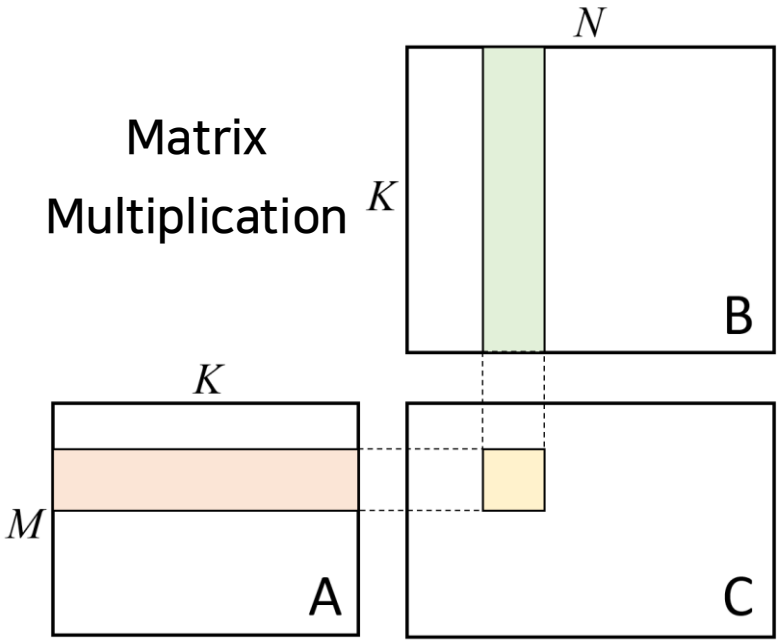
$M = out\_channels$
$K = in\_channels \cdot kernel\_size^2$
$N = out\_height \cdot out\_width$

Proposed Ideas

Characteristics of Diffusion Computation

2. Attention

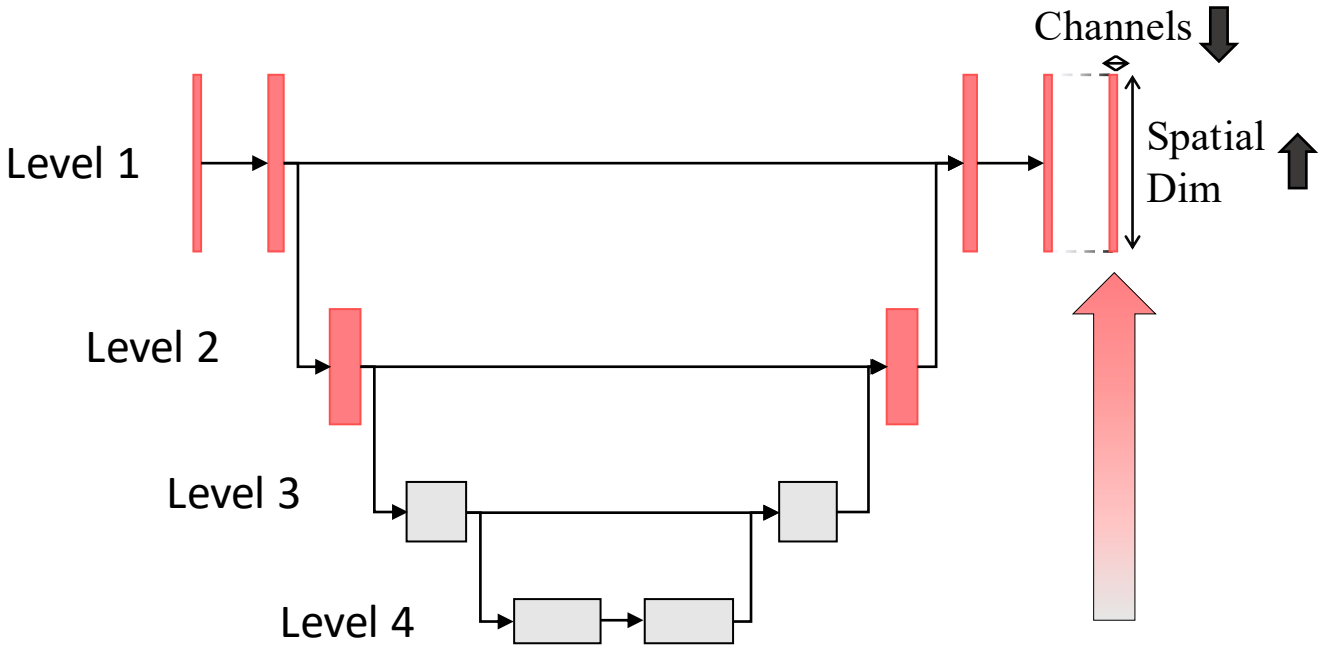




$M = seq\_len$ of the **k**
$K = d_{head}$
$N = seq\_len$ of the **q**

Proposed Ideas

Characteristics of Diffusion Computation



1. Conv2D

Channels ⬇ ⟶ $M\downarrow, K\downarrow$

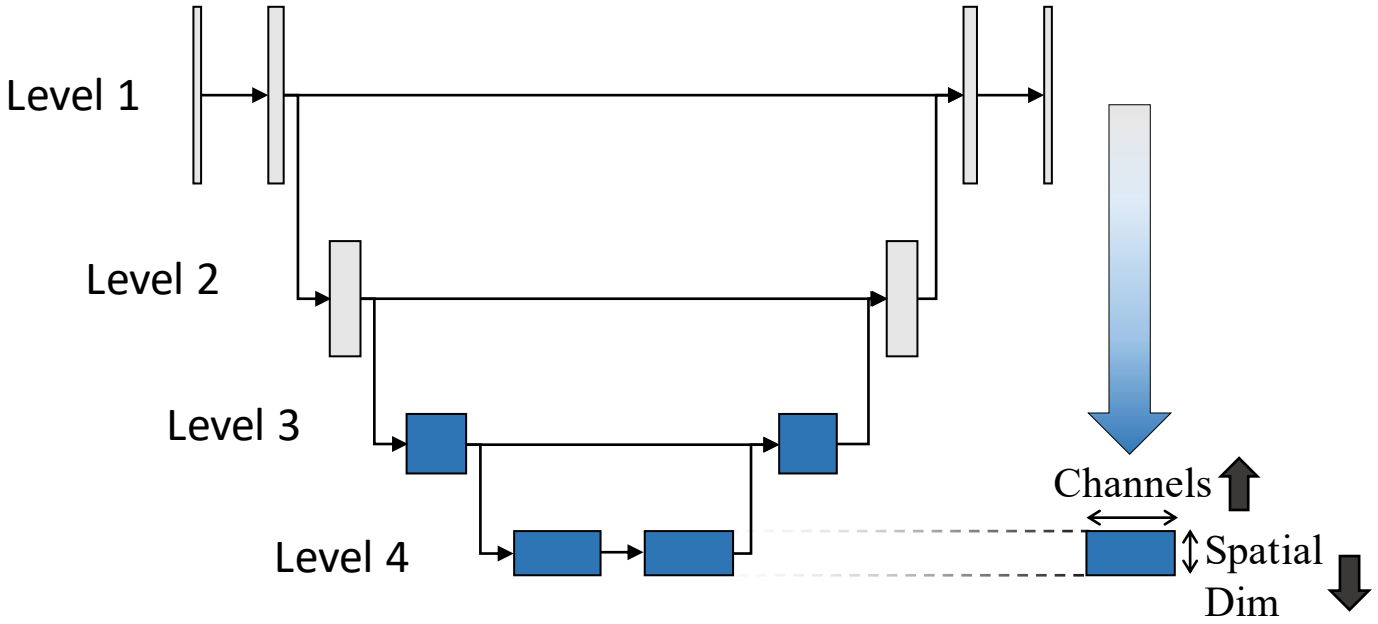
Spatial Dim ⬆ ⟶ $N\uparrow\uparrow$

$$M = out\_channels$$
$$K = in\_channels \cdot kernel\_size^2$$
$$N = out\_height \cdot out\_width$$

2. Attention

- Attention Matrix computation follows the same principle

Characteristics of Diffusion Computation



1. Conv2D

Channels ⬆ ⟶ $M\uparrow, K\uparrow\uparrow$

Spatial Dim ⬇ ⟶ $N\downarrow\downarrow$

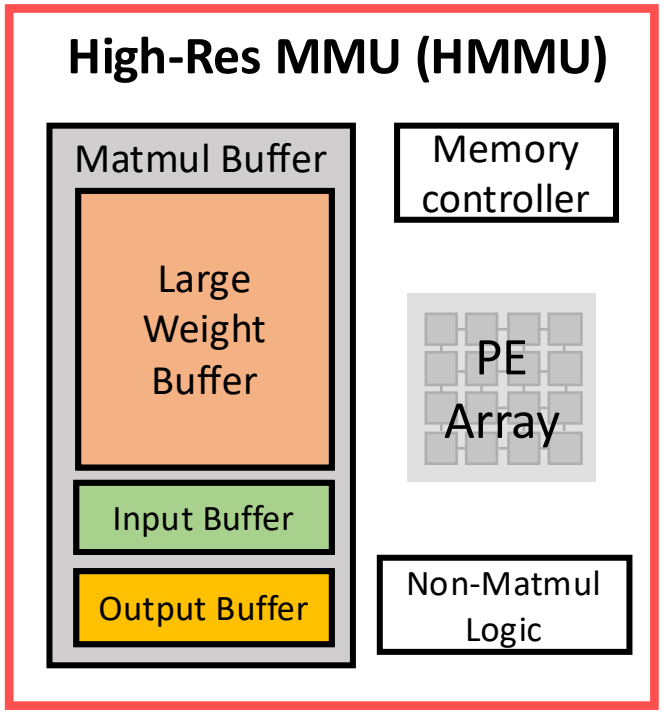
$$M = out\_channels$$
$$K = in\_channels \cdot kernel\_size^2$$
$$N = out\_height \cdot out\_width$$

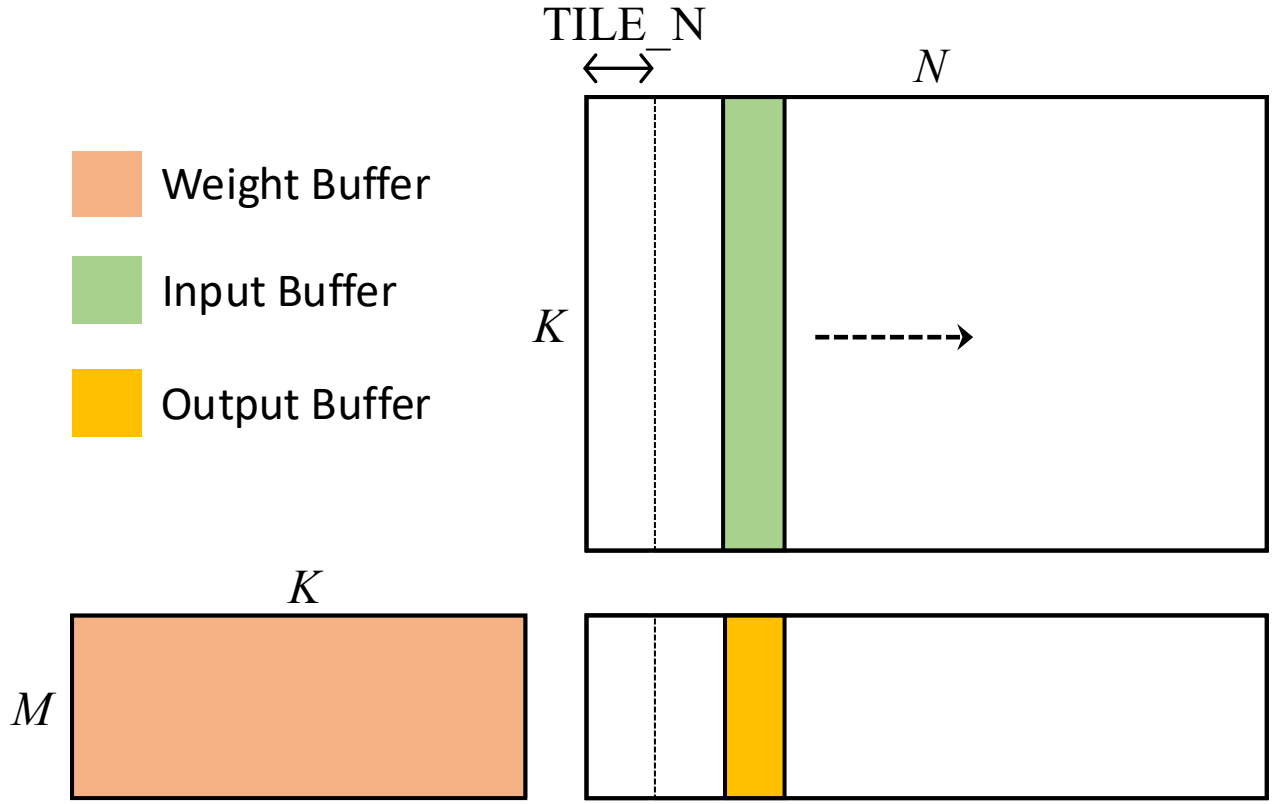
2. Attention

- Attention Matrix computation follows the same principle

Proposed Ideas

Resolution Aware MMU (1) : HMMU



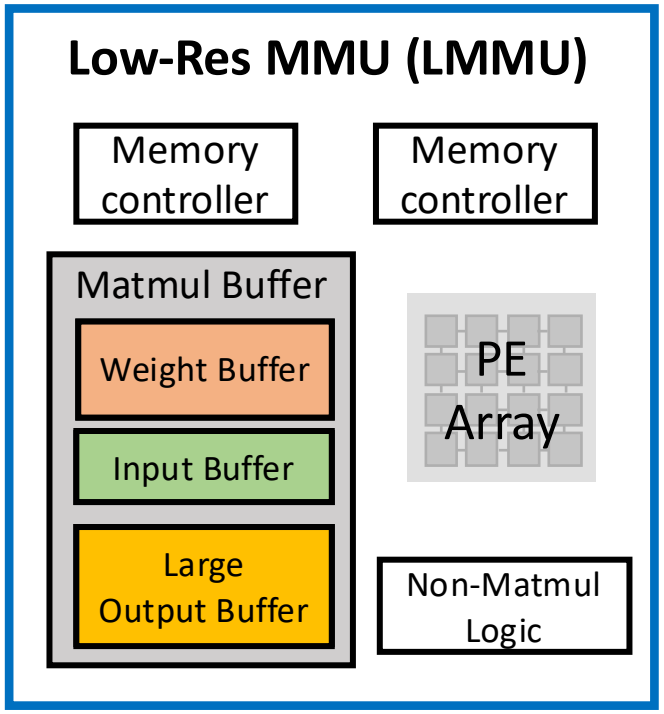


- Large Weight Buffer
- Single Memory controller

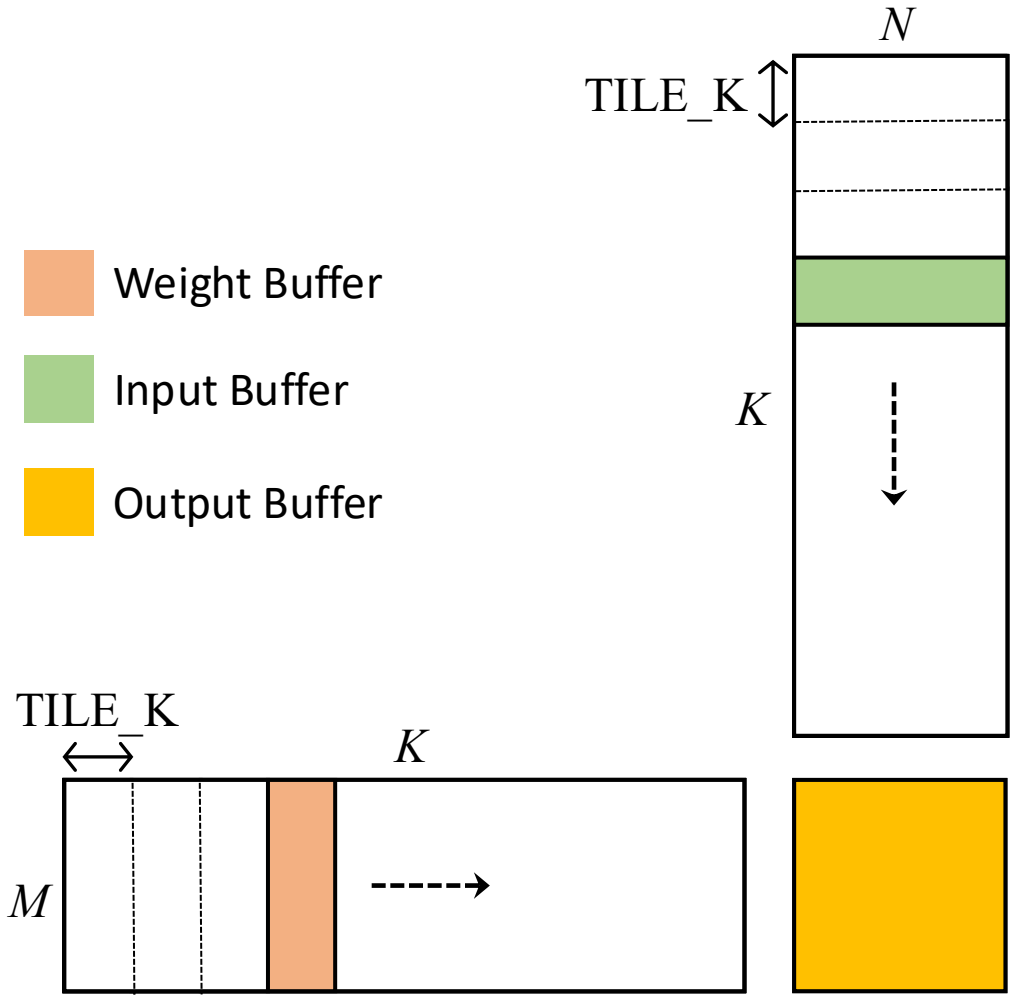
Maximizes data reuse for weight matrices, minimizing DRAM accesses.

Proposed Ideas

Resolution Aware MMU (2) : LMMU



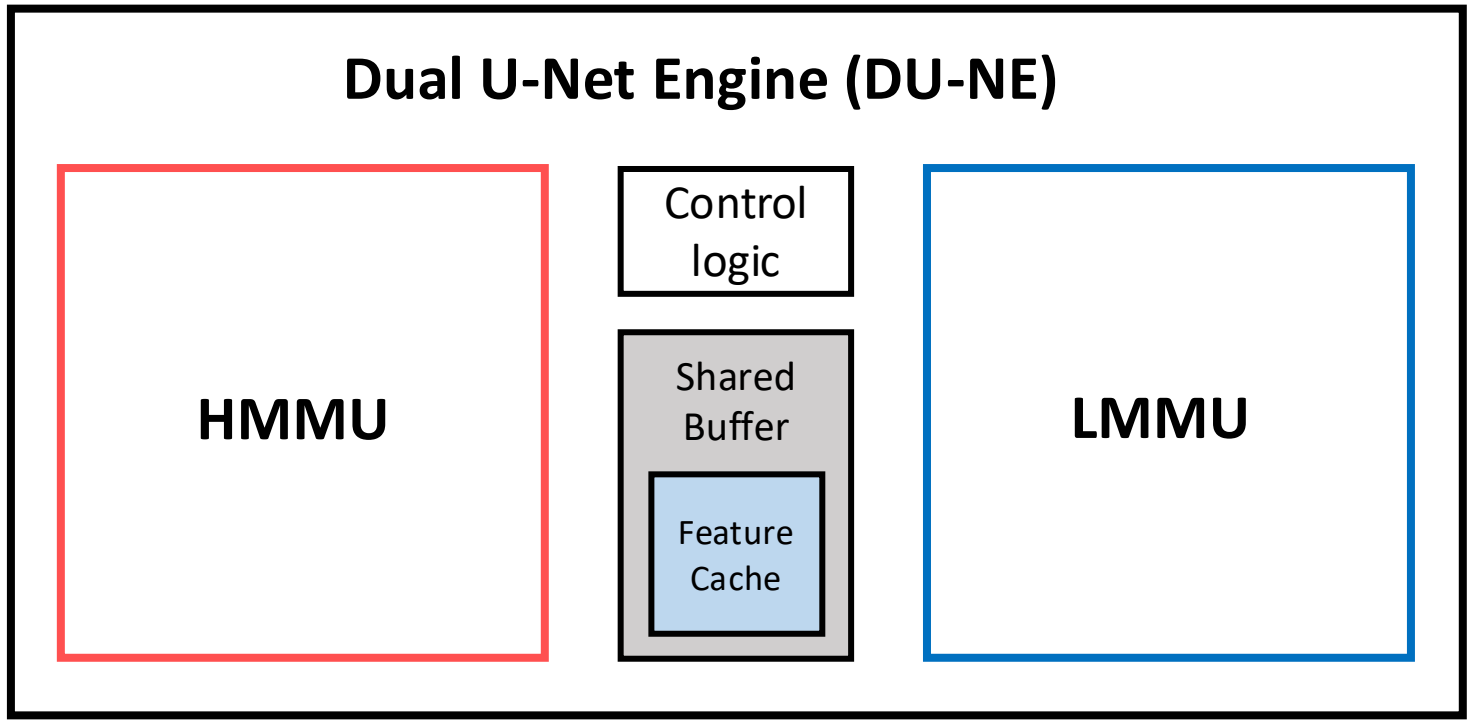
- Small Weight Buffer
- Relatively large Output Buffer
- Two Memory controller



Minimizes DRAM access for output matrices.

Proposed Ideas

Dual U-Net Engine (DU-NE)

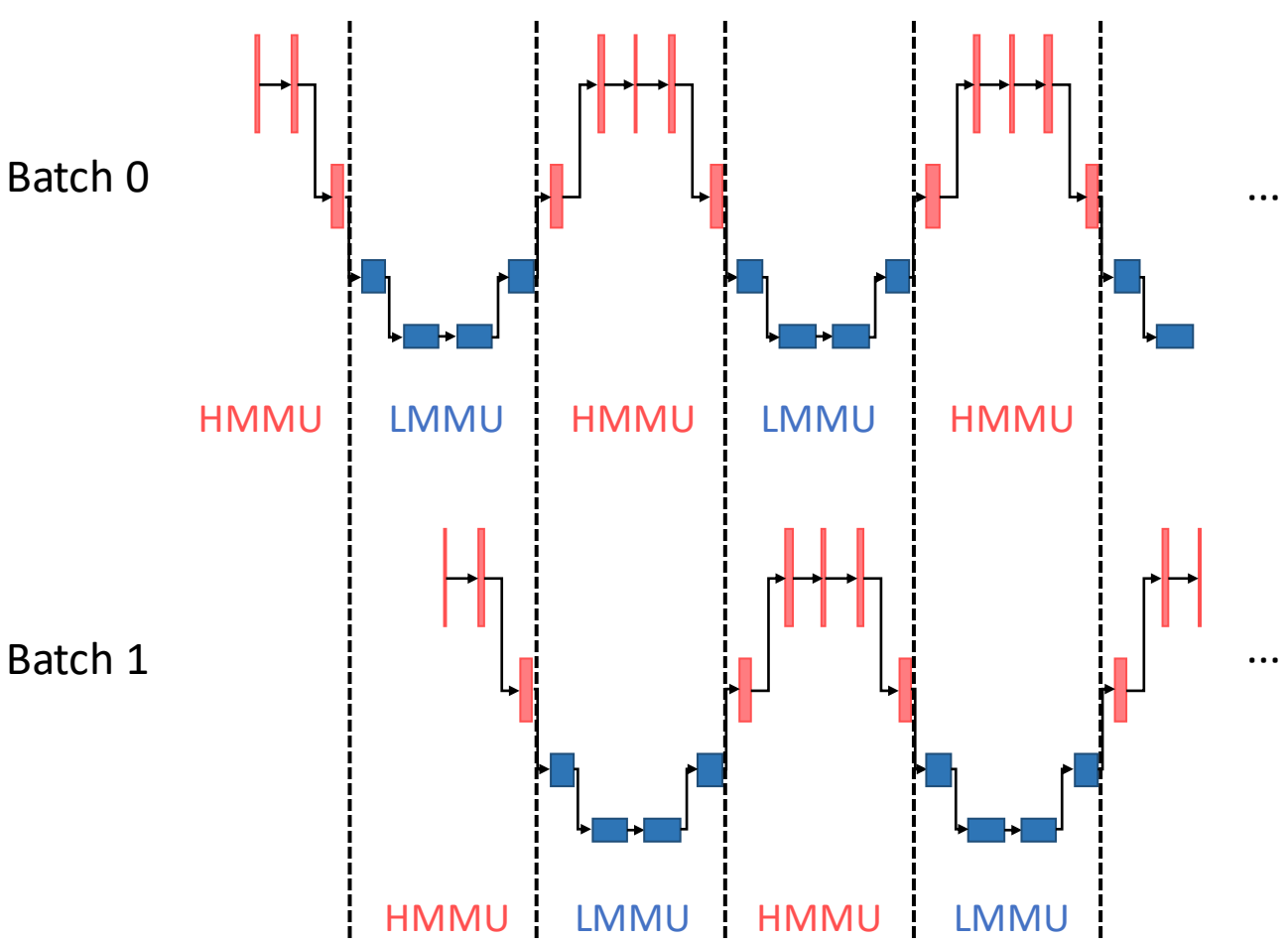


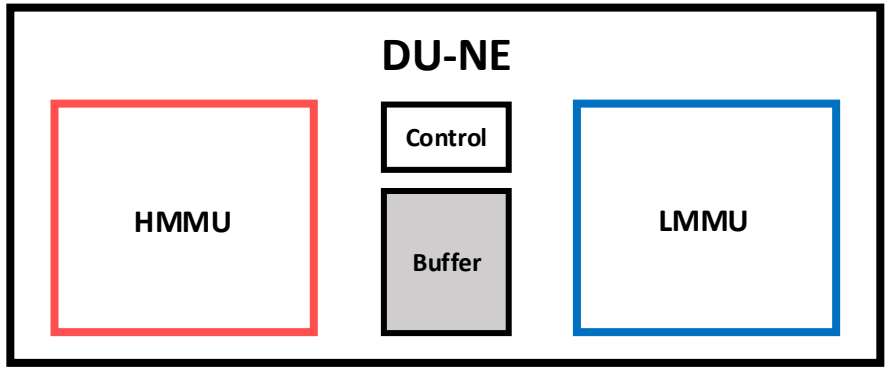
- A core engine for accelerating U-Net model inference.
- Composed of HMMU, LMMU, and a shared buffer.
- Control logic for allocating and scheduling computations across MMUs.

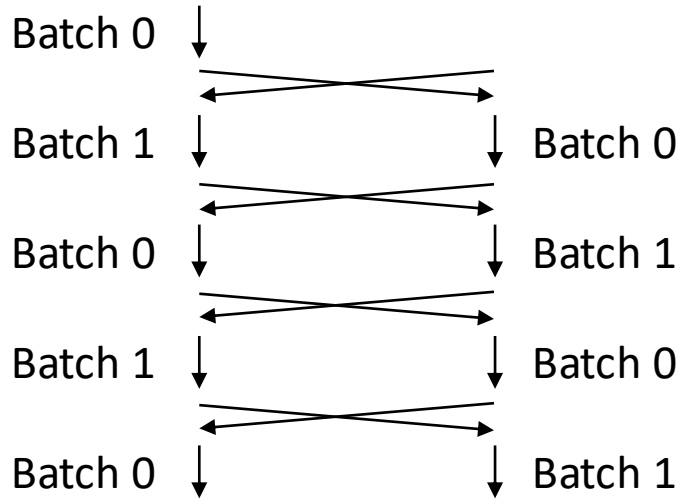
Proposed Ideas

DU-NE Scheduling Method (1)

Multi-Batch & Pipelining Parallelism



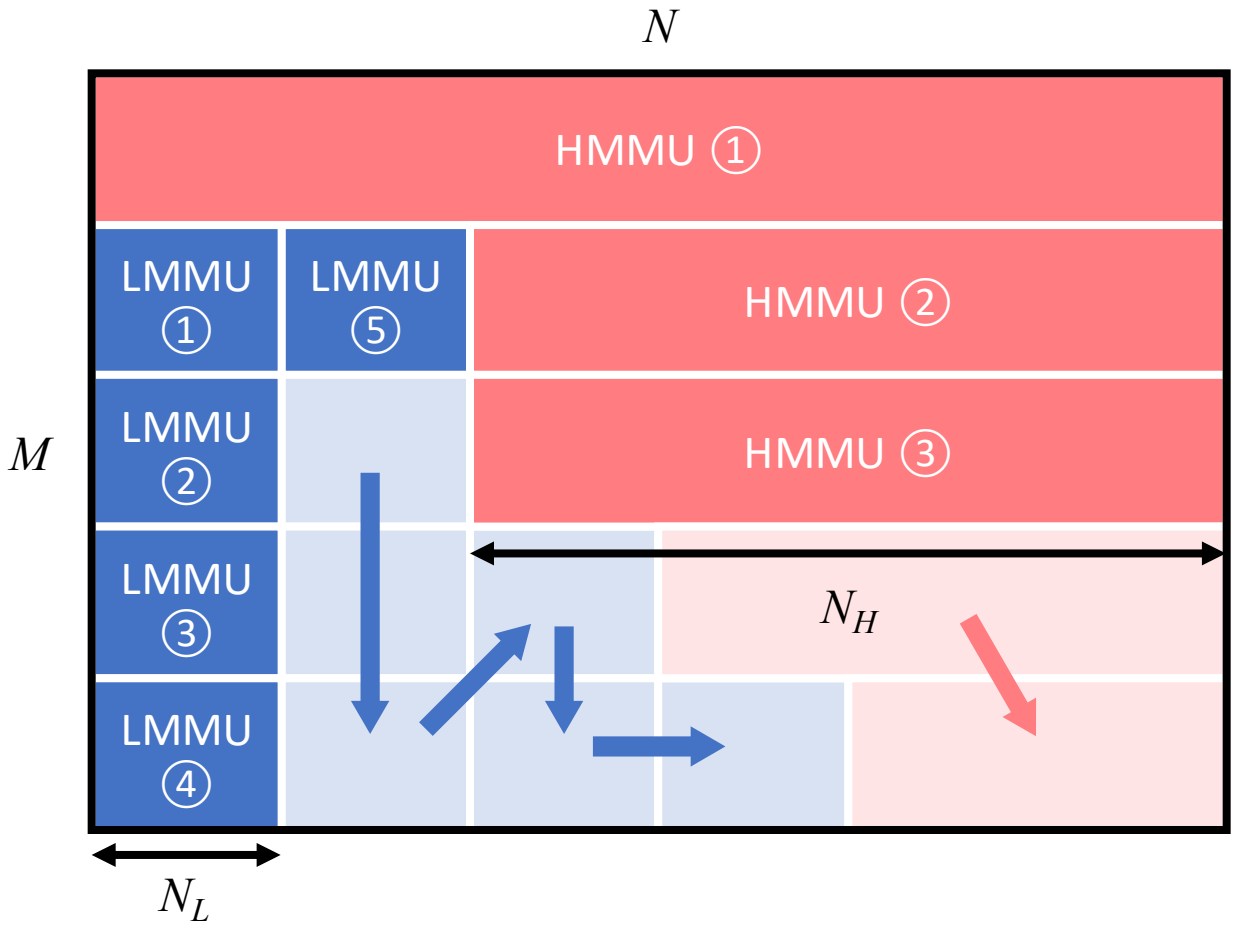




Proposed Ideas

DU-NE Scheduling Method (2)

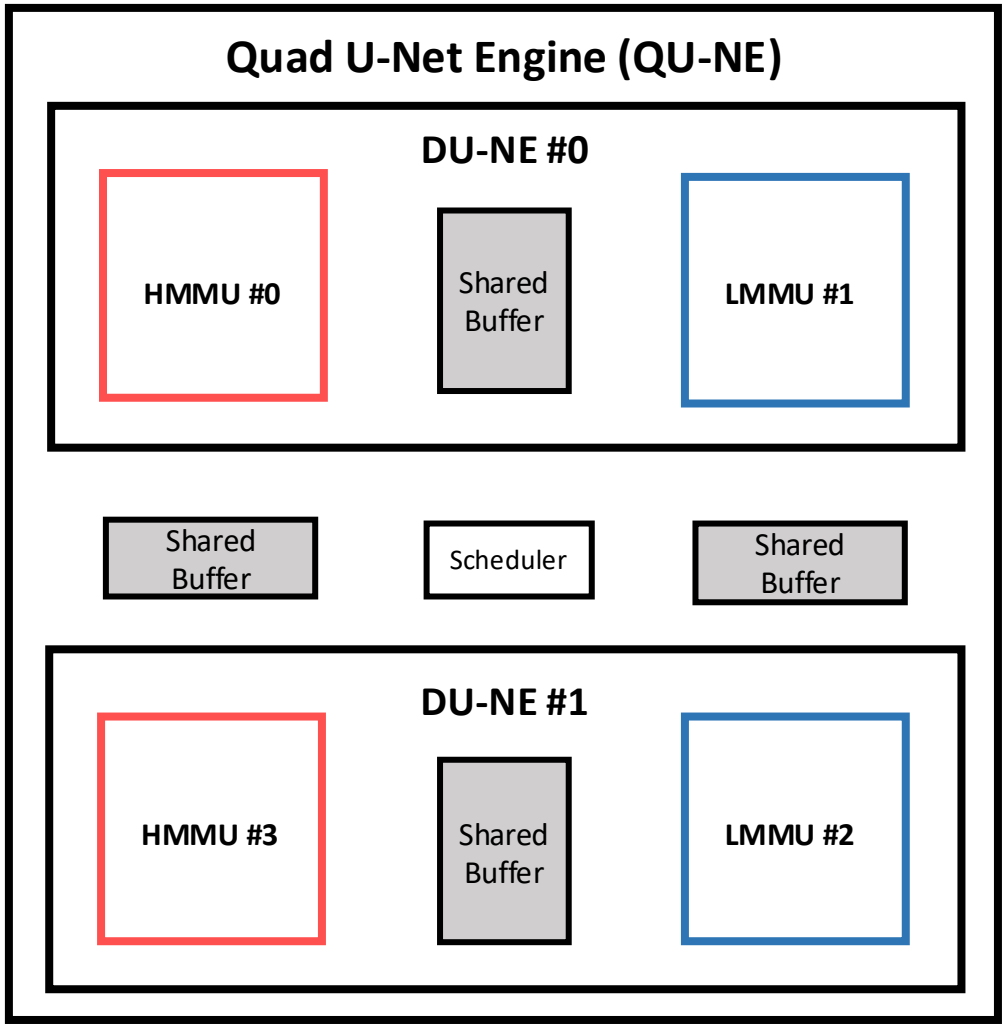
Single Batch & Tensor Parallelism



- Efficiently utilizes HMMU and LMMU by splitting the output matrix based on N.
- HMMU prioritizes large N tiles to minimize overhead.
- LMMU fills smaller N tiles to optimize scheduling.

Proposed Ideas

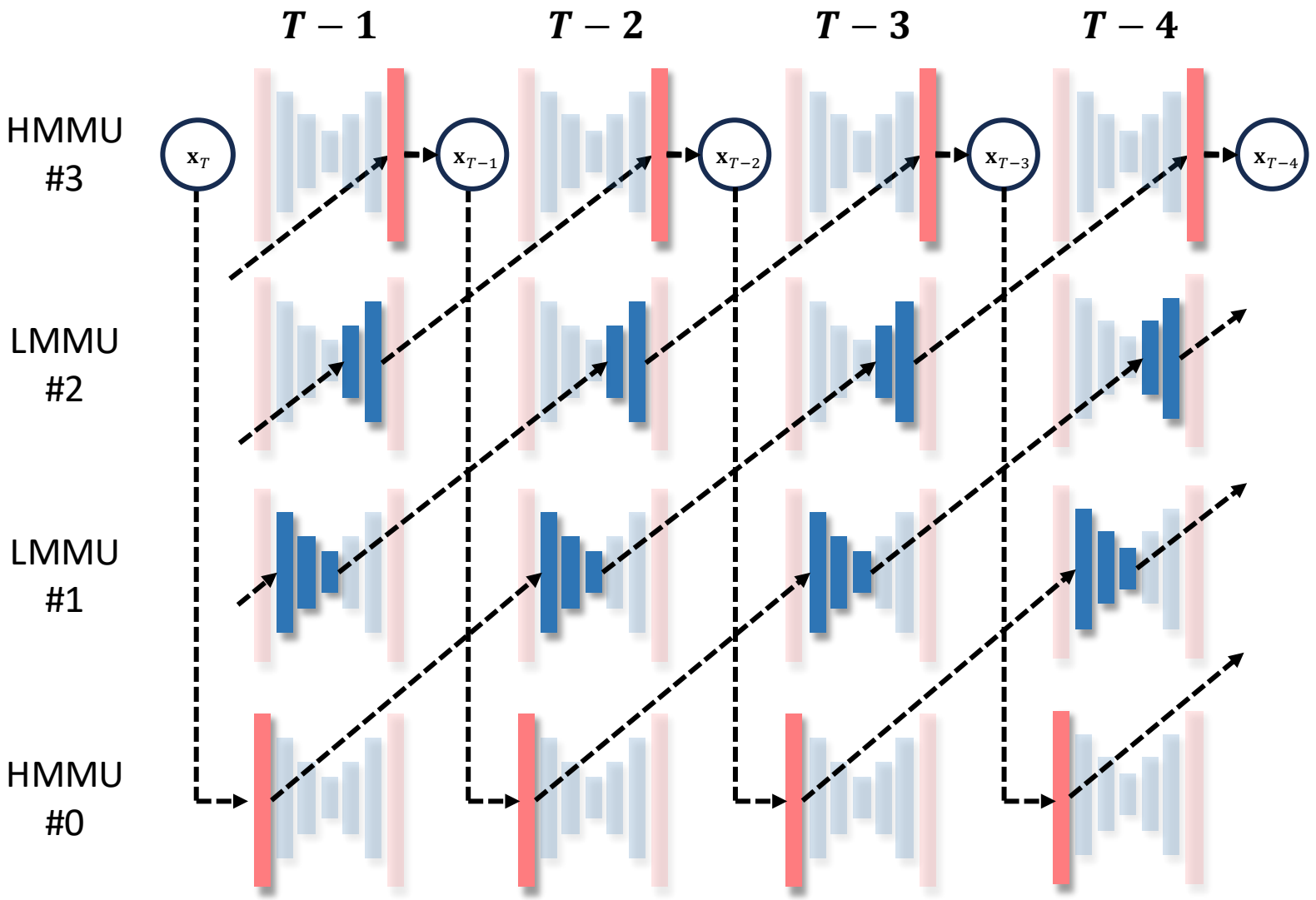
Quad U-Net Engine (QU-NE)

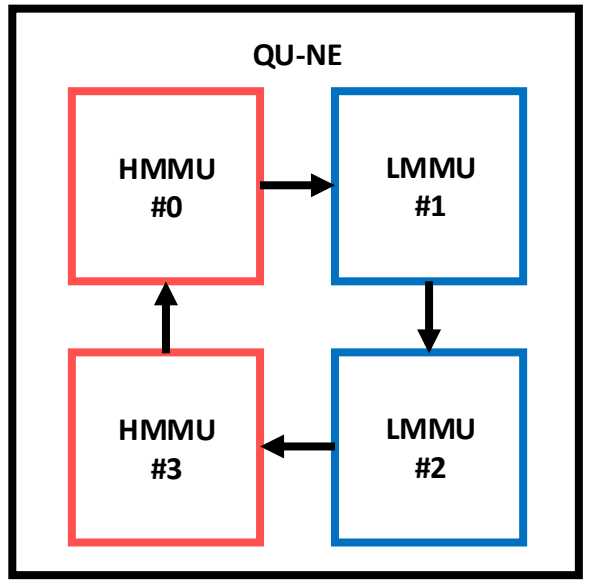


- The largest computation engine for U-Net operations.
- Composed of two DU-NE units connected via a shared buffer.
- MMUs exchange data through adjacent shared buffers.
- Can operate independently as DU-NE units or as an integrated QU-NE system.

Proposed Ideas

QU-NE with AsyncDiff

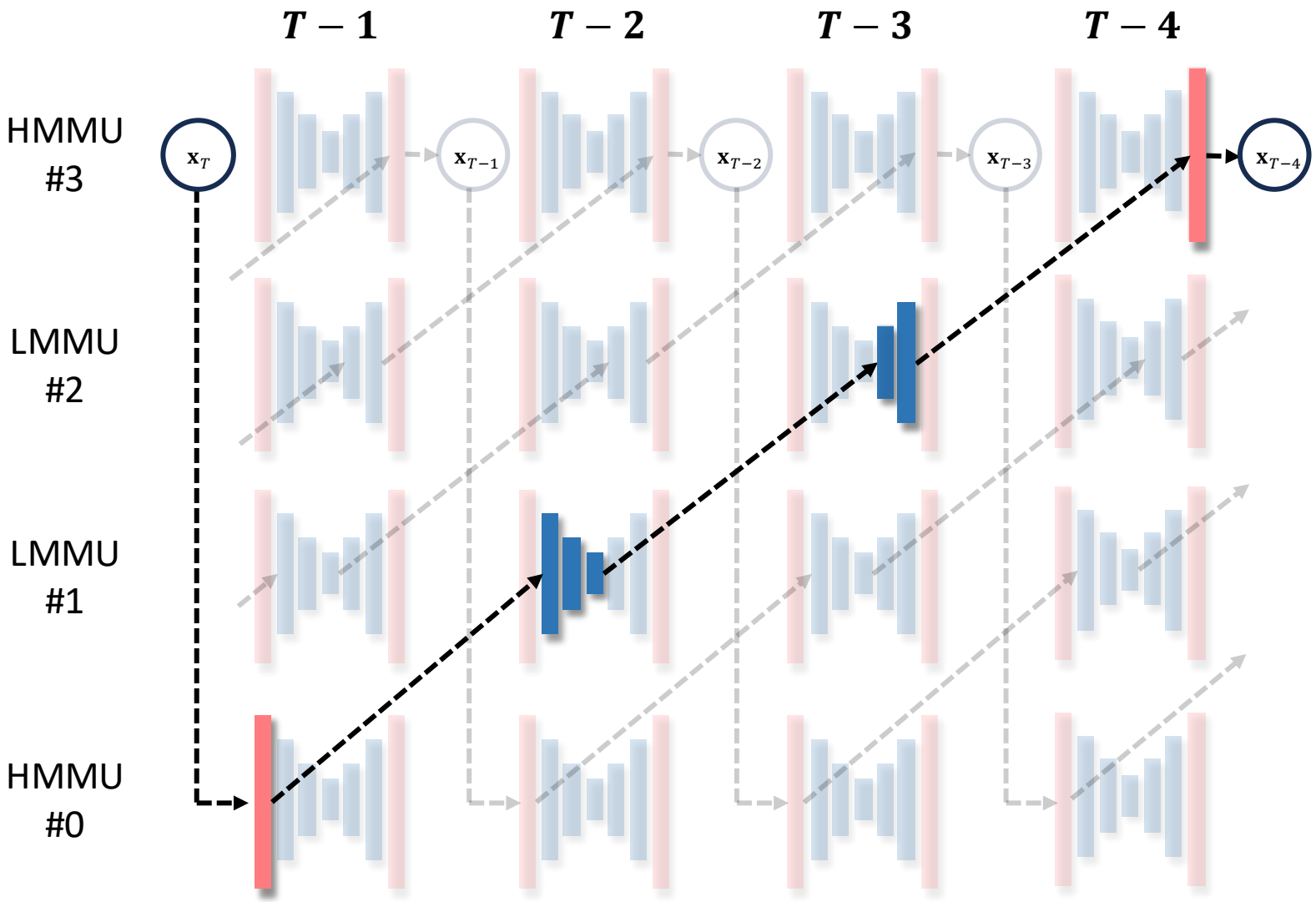


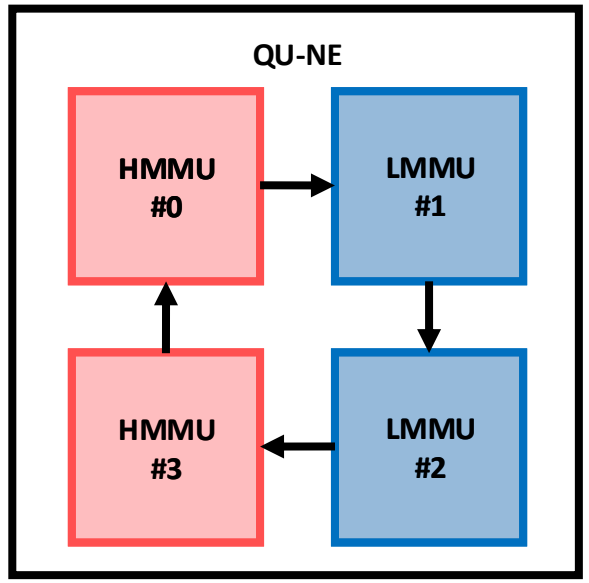


- Enables parallel execution of modules within a timestep.
- Cyclic data flow between MMUs to optimize efficiency.

Proposed Ideas

QU-NE with AsyncDiff

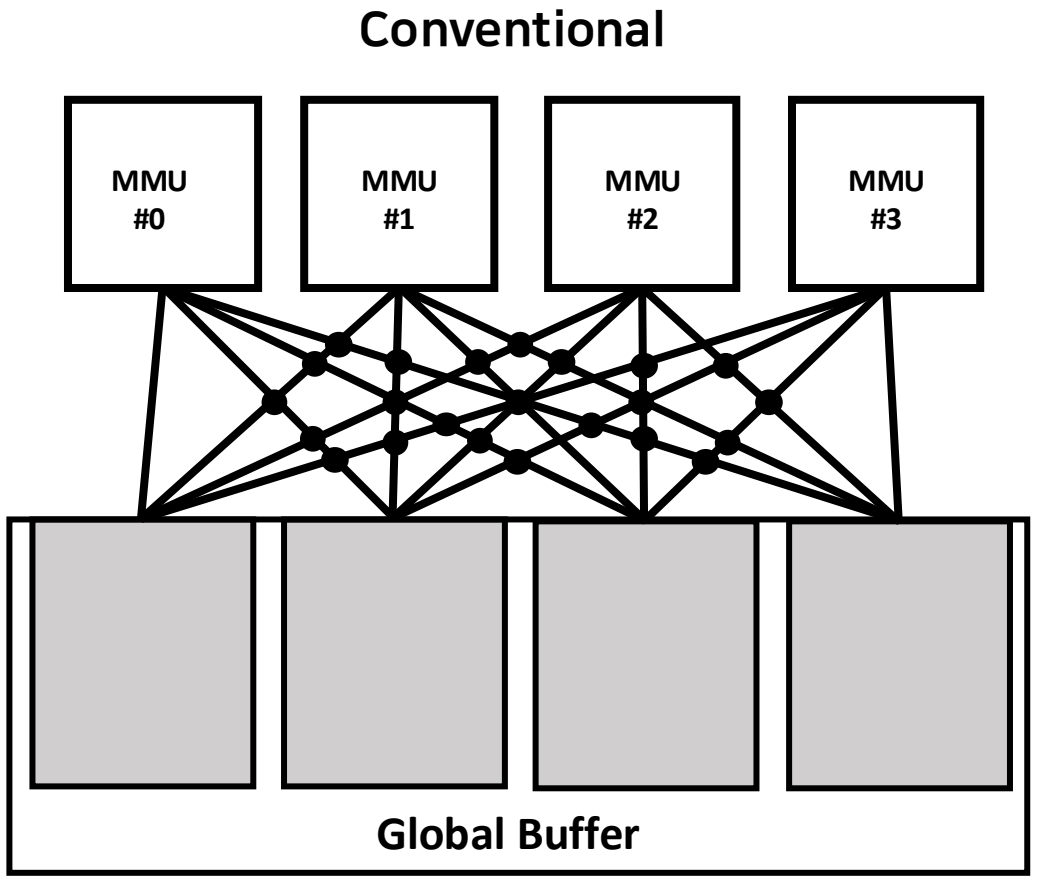


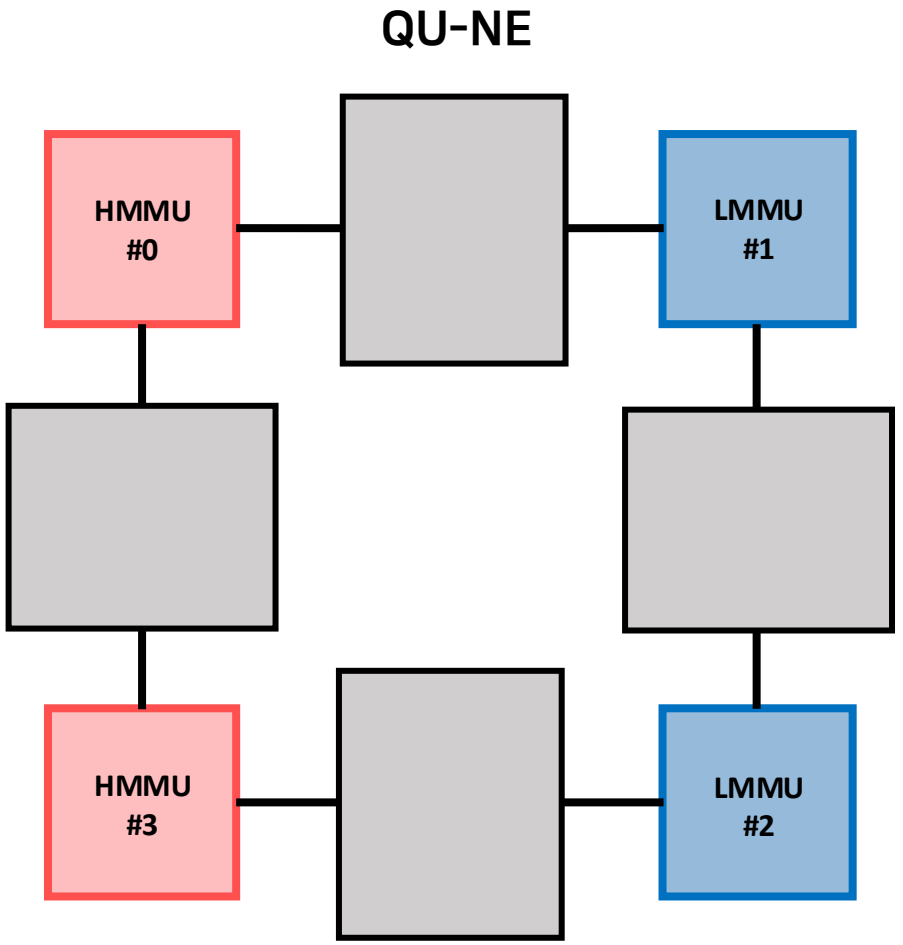


- Enables parallel execution of modules within a timestep.
- Cyclic data flow between MMUs to optimize efficiency.

Proposed Ideas

Efficiency of QU-NE



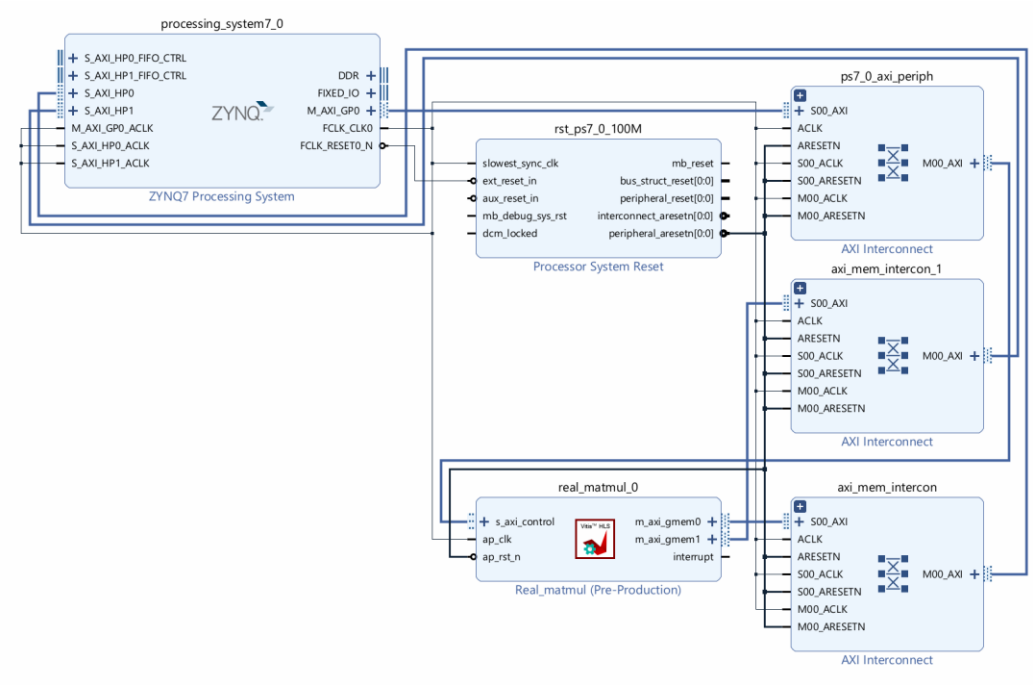


- Complex Wiring, Increased Energy Consumption.
- Complicated Data Flow, Difficult Scheduling.

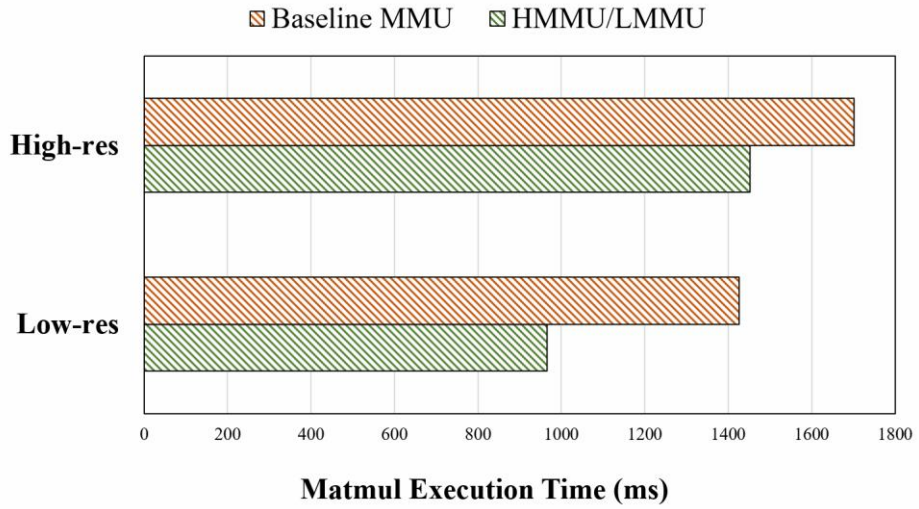
- Simple Wiring, Reduced Power Consumption.
- Efficient Data Flow, Easy Implementation.

Evaluation

Achieved 1.17× speedup for High-Res cases and 1.47× speedup for Low-Res cases compared to the baseline.



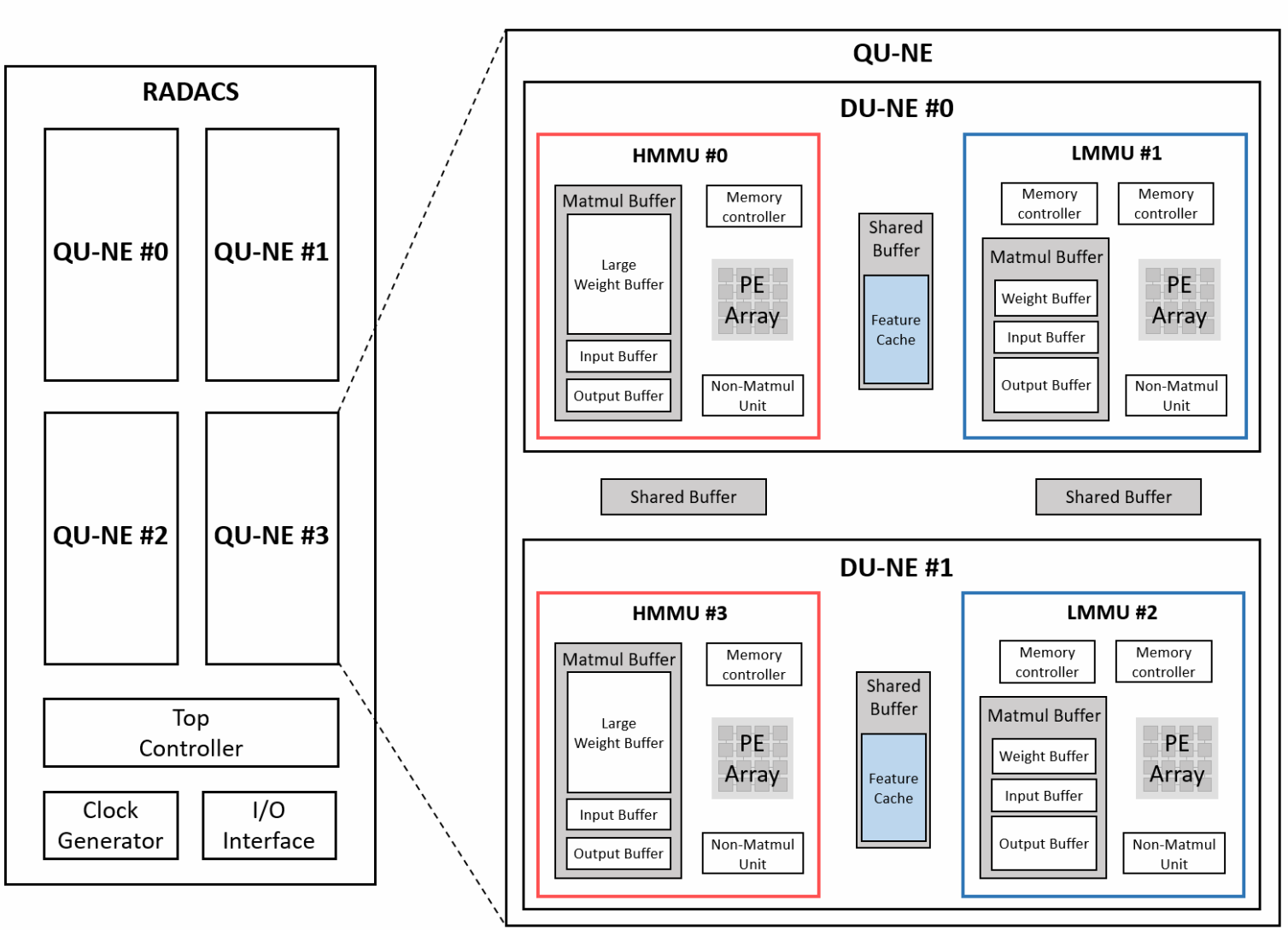
Block Diagram of PS, PL, and AXI Interface

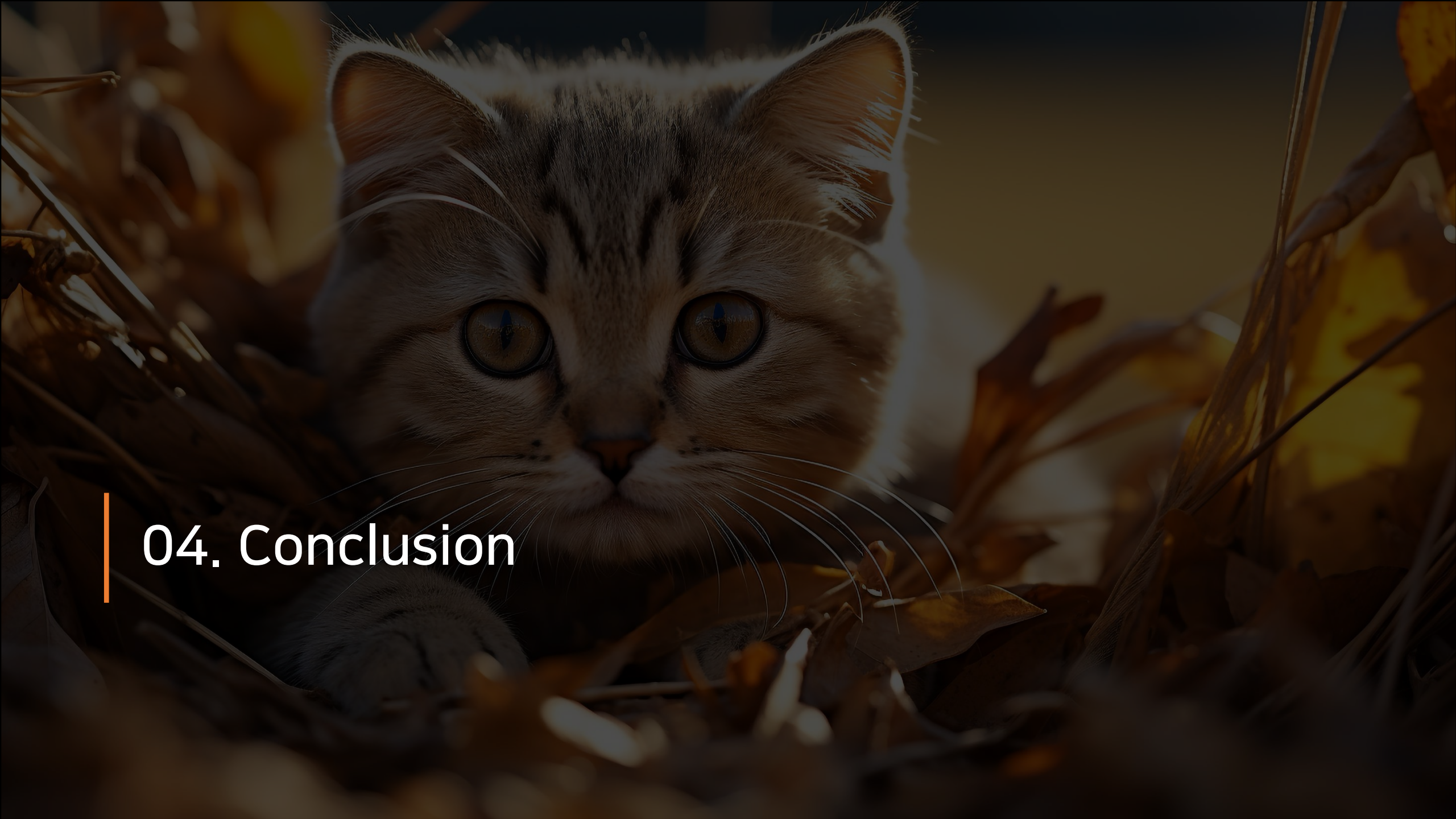


- Compared matrix multiplication execution times on an open-source diffusion model (tiny-random-stable-diffusion-xl).
- Synthesized using Xilinx Zynq-7000 FPGA with Vitis HLS.

Proposed Ideas

RADACS



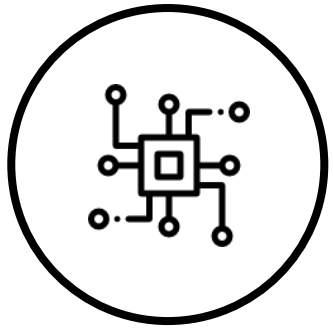


04. Conclusion

Conclusion

Creativity

- The first AI Hardware to integrate the latest staleness characteristics into the design.
- Naturally compatible with the latest algorithms such as DeepCache and AsyncDiff.



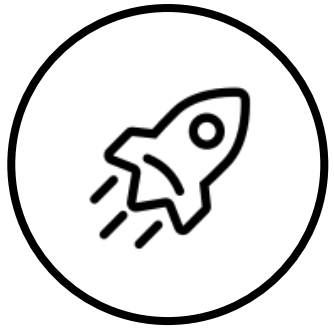
Technical Excellence

- HMMU/LMMU design tailored to the U-Net architecture.
- DU-NE and QU-NE support staleness-based data flow and parallel computation.
- A full-stack solution integrating hardware design and algorithms.

Conclusion

Completeness

- Analyzed M, N, K variations in U-Net's Conv2D and attention operations
- Achieved significant speed up in FPGA-based HLS experiments.
- RTL design expansion and P&R stage planning for precise performance validation.



Expected Impact

- Applicable for image/video processing in edge environments.
- Enables real-time image generation and data processing with improved performance, cost efficiency, and security.
- Useful in robotics for real-time policy generation and other AI applications.

References

[1] X. Ma, G. Fang, X. Wang, “Deepcache: Accelerating diffusion models for free,” in *CVPR*, 2024.

[2] Z. Chen, X. Ma, G. Fang, Z. Tan, et al., “AsyncDiff: Parallelizing diffusion models by asynchronous denoising,” in *NeurIPS*, 2024.

[3] S. Yoo, G. Ko, S. Ham, et al., "Picasso: An Area/Energy-Efficient End-to-End Diffusion Accelerator with Hyper-Precision Data Type," in *ESSERC*, 2024.

[4] Nvidia, "Warp Matrix Functions," [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#alternate-floating-point

[5] C. Chi, Z. Xu, S. Feng, et al, “Diffusion Policy: Visuomotor Policy Learning via Action Diffusion”, in *RSS*, 2023.

[6] F. Wimbauer, B. Wu, E. Schoenfeld, et al., “Cache me if you can: Accelerating diffusion models through block caching,” in *CVPR*, 2024.

[7] X. Yang, X. Wang, “Hash3d: Training-free acceleration for 3d generation.” *arXiv preprint arXiv:2404.06091*, 2024.

[8] J. So, J. Lee, and E. Park, “Frdiff: Feature reuse for exquisite zero-shot acceleration of diffusion models,“ in *ECCV,* 2024


Thank You

Appendix

1. Attention M, K, N

$$M = seq\_len \text{ of the } \mathbf{k}$$
$$K = d_{head}$$
$$N = seq\_len \text{ of the } \mathbf{q}$$

```
q = self.to_q(x)
batch_size, seq_len, _ = q.shape
```

Get batch size and number of elements along sequence axis (`width * height`)

```
self.transformer_blocks = nn.ModuleList(
    [BasicTransformerBlock(channels, n_heads, channels // n_heads, d_cond=d_cond) for _ in range(n_layers)]
)
```

```
80 class BasicTransformerBlock(nn.Module):
```

```
85     def __init__(self, d_model: int, n_heads: int, d_head: int, d_cond: int):
```

- The sequence length of q is proportional to the spatial dimension of the feature, resulting in a larger value in high-resolution cases.
- For k, when the input is a text condition, it has a fixed embedding dimension regardless of the feature.
- *d_{head} is obtained by dividing the number of channels by the number of heads in multi-head attention. Since the number of heads is fixed, d_{head} is proportional to the number of channels.*

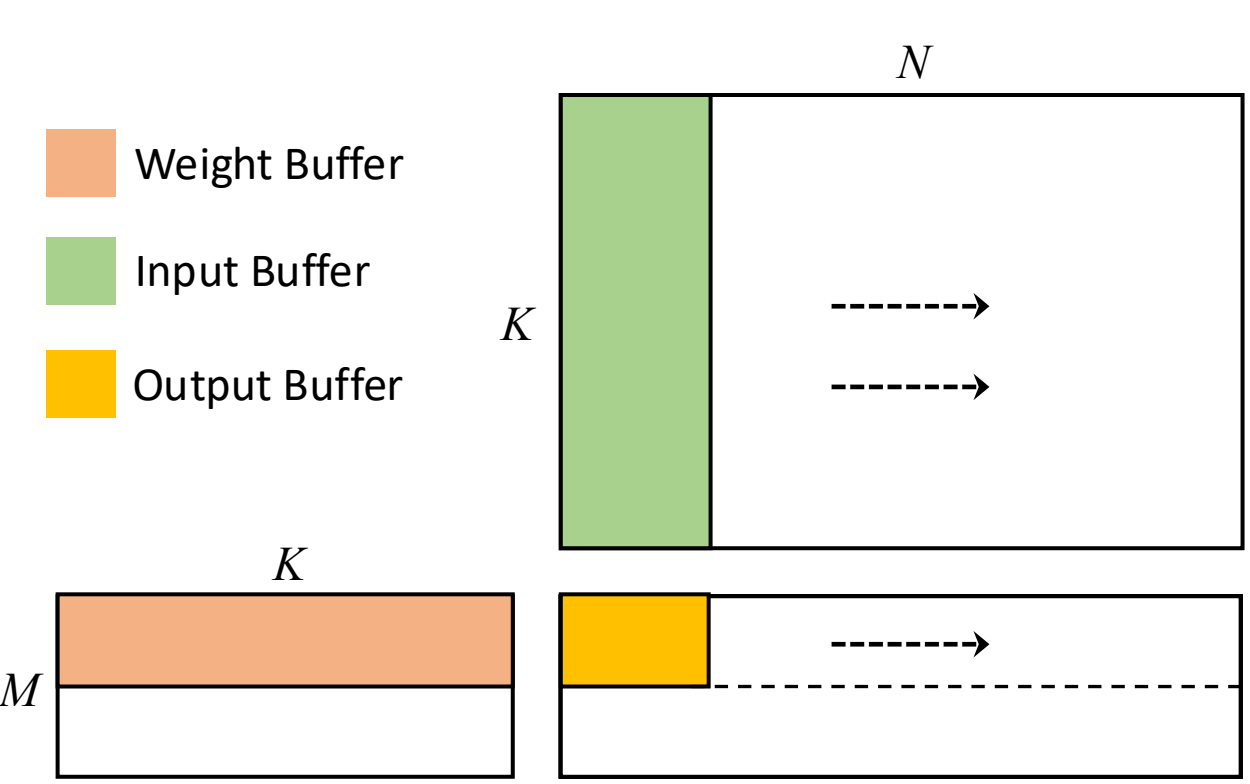
High-res case → $N \uparrow\uparrow$

Low-res Case → $K \uparrow\uparrow$

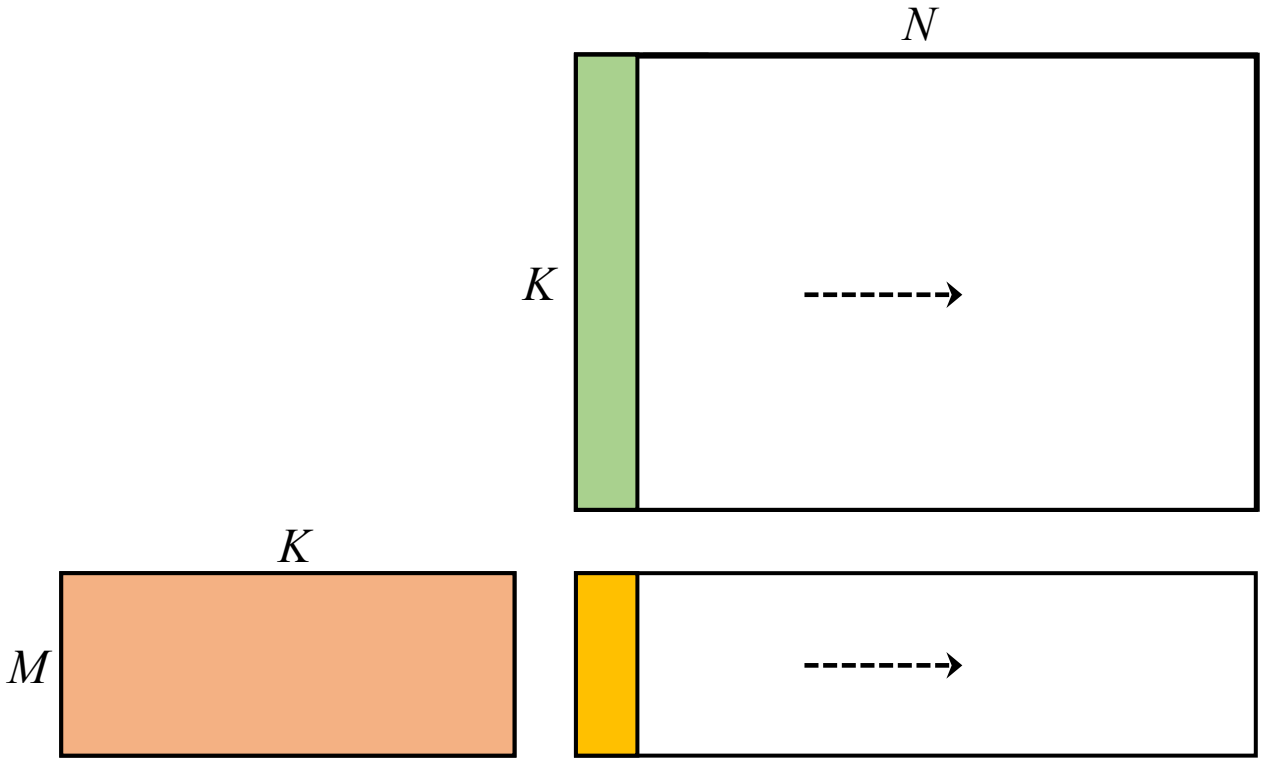
https://nn.labml.ai/diffusion/stable_diffusion/model/unet_attention.html

Appendix

2. Matmul DRAM access breakdown



Dram access : M x K + 2 x K x N



Dram access : M x K + K x N